CROCODILE Benutzerhandbuch

Der Cisco Router Configuration Diligent Evaluator

Autoren:

Holger Peine
Thomas Schwenkler
Reinhard Schwarz
Kai Simon

IESE Report-Nr. 119.03/D
Version 3.0
November 2003

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer Gesellschaft. Es überträgt innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach
Sauerwiesen 6
67661 Kaiserslautern

Vorwort

Im folgenden beschreiben wir das Prüfwerkzeug CROCODILE. Das Werkzeug analysiert Routerkonfigurationen und identifiziert darin enthaltene potentielle Sicherheitsschwachstellen. Damit unterstützt CROCODILE den Anwender darin, Rechnernetze einer Sicherheitsrevision zu unterziehen.

Zielsetzung und Leserkreis des Handbuchs

Das vorliegende Handbuch richtet sich vornehmlich an den Anwender. Es gibt einen Überblick über die Handhabung des Werkzeugs und dessen Prüf- und Unterstützungsfunktionen. Darüber hinaus soll es den CROCODILE-Nutzer mit der Werkzeugstruktur und den grundlegenden Programmabläufen so weit vertraut machen, daß dieser kleinere Anwendungsprobleme selbständig meistern kann.

Das Handbuch richtet sich auch an die Noch-nicht-Anwender des Werkzeugs — potentielle Interessenten also, die unschlüssig sind, ob CROCODILE ein geeigneter Baustein für ihre Sicherheitslösung sein könnte. Dem unentschlossenen Leser wollen wir hinreichende Informationen über die Möglichkeiten und Grenzen des Werkzeugs zur Hand geben, damit er sich guten Gewissens für oder gegen den CROCODILE-Ansatz entscheiden kann.

Potentielle Entwickler, die das Werkzeug selbst modifizieren oder erweitern wollen, sollten das Benutzerhandbuch als einführendes Dokument lesen, ehe sie Hand an das System legen. Mit Rücksicht auf reine Anwender haben wir bei der Darstellung bewußt darauf verzichtet, allzu tief in die Details der Realisierung einzudringen. Insbesondere haben wir im vorliegenden Dokument nur wenige Programmfragmente vorgestellt, um Leser ohne PERL-Kenntnisse nicht zu überfordern. Der Anwender kann die entsprechenden Textpassagen notfalls getrost überspringen, ohne dadurch den roten Faden zu verlieren. Dessen ungeachtet findet der Entwickler im folgenden viele nützliche Informationen zur Werkzeug-Implementierung.

Notwendige allgemeine Vorkenntnisse

Unser Bestreben war es, ein Werkzeug zu konzipieren, das einfach installierbar, intuitiv bedienbar und möglichst kompatibel zu unterschiedlichsten Ausführungsplattformen ist. Computergrundkenntnisse sollten ausreichen, um das

Copyright © Fraunhofer IESE 2003

Werkzeug erfolgreich in Betrieb zu nehmen. Da der typische Anwender von CROCODILE mit Routerkonfigurationen befaßt ist, können wir grundlegende Fertigkeiten im Umgang mit Computern und Software wohl voraussetzen.

Das Werkzeug wird per Kommandozeile — meist aus einer Benutzer-Shell, einem Konsolenfenster oder dergleichen — gestartet. Der Anwender sollte daher mit der Kommandozeilen-Schnittstelle seines Betriebssystems vertraut sein. Unsererseits sind wir nicht auf eine bestimmte Plattform fixiert: Das Werkzeug wurde teils unter Unix-Dialekten (Solaris, Linux), teils unter Windows-Varianten entwickelt.

Prüfergebnisse werden für die Darstellung in einem HTML-Browser aufbereitet. Da HTML-Dokumente von verschiedenen Browser-Typen unterschiedlich dargestellt werden, ist es mitunter erforderlich, einige Browser-Einstellungen nachzujustieren. Der Anwender sollte wissen, wie er zum Beispiel Schriftgrößen verändern und «HTML Frames» oder «Cascading Style Sheets» bei seinem Browser aktivieren kann. Im Internet-Zeitalter erachten wir dies als zumutbar. Weitere Darstellungsparameter — vor allem Farbzuordnungen — lassen sich aus CROCODILE heraus konfigurieren. Das Handbuch beschreibt die dazu erforderlichen Schritte.

Notwendige Router-Kenntnisse

CROCODILE analysiert Routerkonfigurationen, genauer: Konfigurationsbeschreibungen für Router, die unter dem Internet Operating System (IOS) der Firma Cisco Systems laufen. Es ist daher selbstverständlich, daß der Anwender mit Routing-Problemen und mit IOS vertraut sein muß, um von den Analysen des Werkzeugs zu profitieren. Die Durchführung geeigneter Maßnahmen in Reaktion auf das Prüfergebnis erfordert in jedem Falle Routing-Fachwissen.

Das Werkzeug verwendet teils feste, teils frei konfigurierbare Prüfkriterien. Um eigene Prüfkriterien zu formulieren, muß der Anwender über ausreichende Router- und Sicherheitskenntnisse verfügen. Da Sicherheitsanforderungen je nach dem Einsatzkontext sehr verschieden sind, kann CROCODILE keine universell sinnvolle Default-Konfiguration vorhalten: Die mitgelieferten Konfigurationseinstellungen dienen hauptsächlich der Illustration und liefern voraussichtlich einige sinnlose Analyseergebnisse. Die benutzerdefinierbaren Prüfregeln bedürfen daher der Anpassung an die lokalen Gegebenheiten. Das Handbuch erläutert die notwendigen Schritte an passender Stelle.

Modifikation und Erweiterung des Werkzeugs

CROCODILE wurde als erweiterbares Framework konzipiert. Das bedeutet, daß der Funktionsumfang des Werkzeugs nicht festgelegt ist, sondern leicht ergänzt oder

Copyright © Fraunhofer IESE 2003

verändert werden kann. Der Anwender kann bei Bedarf selbst als Entwickler tätig werden, indem er zusätzliche — oder gänzlich andere — Prüfmodule konzipiert und in das Framework einsetzt.[1] Das Benutzerhandbuch liefert hierzu grundlegende Einsichten — einen Startpunkt, um die Struktur und Funktionsweise des Werkzeugs zu ergründen. Es kann allerdings ein Entwicklerhandbuch oder eine Entwicklerschulung durch Fraunhofer IESE nicht ersetzen.

Um eigene Werkzeugerweiterungen vorzunehmen, ist Programmiererfahrung in der Programmiersprache PERL unerläßlich. Die Grundzüge der Werkzeugarchitektur und das Konstruktionsprinzip von Prüfmodulen vermittelt das Benutzerhandbuch. Ausreichende Programmierkenntnisse vorausgesetzt, können die verfügbaren Prüfmodule als Vorlage für eigene Weiterentwicklungen dienen, sofern der Anwender eine Quelltext-Lizenz von CROCODILE erworben hat. (Das Werkzeug wird auch in als «Obfuscated Version» mit verschleiertem Quelltext angeboten.)

Neuerungen in der CROCODILE Version 3.0

Gegenüber den Vorgängerversionen bietet die Version 3.0 des Werkzeugs vor allem die folgenden Verbesserungen:

- **Stapelverarbeitungsmodus**
  Ein neues Frontend für Stapelverarbeitung ermöglicht die unbeaufsichtigte Reihenanalyse vieler Konfigurationsdateien in Serie. Die Einzelergebnisse werden in vergleichender Darstellung tabelliert.

- **CSV-Ausgabeschnittstelle**
  Die Ergebnisse eines Stapelverarbeitungslaufs werden wahlweise auch im Comma-separated Vector (CSV) Format ausgegeben und können so leicht mit anderen Werkzeugen — etwa MS Excel — weiterverarbeitet werden.

- **Ausdrucksstärkere Prüfregel-Beschreibungssprache**
  Zur Formulierung eigener Prüfregeln steht dem Anwender nun eine wesentlich erweiterte Regelsyntax zur Verfügung. Prüfkriterien für mehrere Konfigurationsklauseln lassen sich damit logisch verknüpfen, und auch die bedingte Anwendung von Prüfregeln ist spezifizierbar.

- **RAT Emulation**
  CROCODILE ist in der Lage, Original-Prüfregelsätze für das Router Audit Tool (RAT), einem anderen IOS-Prüfwerkzeug, zu verarbeiten und RAT-konforme

1 Wahlweise bieten wir an, solche Ergänzungen oder Anpassungen im Kundenauftrag durchzuführen.

Copyright © Fraunhofer IESE 2003

Prüfberichte zu generieren. Dies ermöglicht RAT-Nutzern den einfachen Umstieg auf das wesentlich leistungsfähigere CROCODILE.

- **Gesteigerte Rechenleistung**
  Obwohl die neue Version an Umfang und Leistungsvermögen erheblich gewachsen ist, konnte die Ausführungsgeschwindigkeit der Prüfläufe deutlich — in einigen Anwendungen um mehr als das Vierfache — gesteigert werden. Die Beschleunigung kommt vor allem dem Stapelverarbeitungsbetrieb zugute. Im interaktiven Betrieb erhöht sie den Bedienkomfort, insbesondere auf leistungsschwachen Rechnersystemen.

Kontakt

Ein Handbuch kann nicht alle Fragen klären, die sich beim Einsatz von CROCODILE ergeben werden. Wir hoffen jedoch, zumindest ausreichende Grundlagen zu schaffen, damit sich der Anwender die subtilen Details des Werkzeuggebrauchs eigenständig erschließen kann.

Um künftigen Anwendern den Umgang mit CROCODILE zu erleichtern, freuen wir uns über jeden Verbesserungsvorschlag aus dem Anwenderkreis. Anmerkungen zum Werkzeug oder der Werkzeugbeschreibung können an folgende Adresse gerichtet werden:

iese_its@iese.fraunhofer.de

Ein entsprechender Link — 'Contact' — ist in die Hypertext-Ausgabedokumente des Werkzeugs integriert.

Copyright © Fraunhofer IESE 2003

Inhaltsverzeichnis

Vorwort **v**

1 Problemstellung und Lösungsansatz **1**

1.1 Netzsicherheit durch sichere Router-Konfiguration 1

1.2 Werkzeugunterstützung für Router-Sicherheitsrevisionen 1

1.3 Sicherheitsprüfkriterien 2

1.4 Gliederung des Handbuchs 3

2 Anwendung des Prüfwerkzeugs im Überblick **5**

2.1 Systemvoraussetzungen 5

2.2 Installation und Deinstallation des Prüfwerkzeugs 6

2.3 Auswahl der Eingabedaten 6

2.4 Probelauf 7

2.5 Darstellung der Prüfergebnisse 9

2.6 Laufzeitoptionen und Help-Funktion 11

2.7 Die nächsten Schritte … 12

3 Architektur und Funktionsprinzip **15**

3.1 Übersicht 15

3.2 Parser-Modul 20

3.3 Prüfmodule 28

3.4 Befunddatenbank 32

3.5 Ausgabe des Analysebefunds 35

Copyright © Fraunhofer IESE 2003

3.6 Unterstützungsmodule 41
3.7 Verzeichnisstruktur 42

4 Prüfmodule 45
4.1 Auswahl von Prüfmodulen zur Ausführung 45
4.2 Modul 'CompoundPatterns' 46
4.3 Modul 'IngressEgress' 56
4.4 Modul 'Connectivity' 64
4.5 Modul 'SNMP' 70
4.6 Modul 'NTP' 72
4.7 Modul 'AAA' 75
4.8 Modul 'Logging' 78
4.9 Modul 'Passwords' 81

5 Emulation des Router Audit Tools (RAT) 85
5.1 Emulationsmodul 'RATemulation' 85
5.2 RAT-Konformität 87
5.3 Konfiguration des Emulationsmoduls 88
5.4 Kennzahlberechnung (CIS Benchmark und CROCODILE Score) 90
5.5 Prüfreports 91
5.6 Regelbeschreibungen 93
5.7 Bewertung 94

6 Stapelverarbeitung (Batch Processing) 97
6.1 Verwendung von CROCODILE im Stapelverarbeitungsmodus 97
6.2 Konfigurierbare Ausführungsparameter (Options) 101
6.3 Auswahl von Prüfregelsätzen 103
6.4 Maschinenlesbare Ergebnisformate 104

Copyright © Fraunhofer IESE 2003

6.5 Protokollierung der Stapelverarbeitung 105

7 Hilfswerkzeuge (Utilities) 107

7.1 Durchlässigkeitsanalysen an ACLs und Interfaces mittels 'blackwhite' 107

7.2 Ergebnisaufbereitung mittels 'make_html' 111

7.3 Generierung von URLs auf die IOS Command Reference mittels 'linkfilter' 112

8 Erstellen eigener Prüfmodule 115

8.1 Recherche 115

8.2 Analyse 116

8.3 Implementieren der Prüflogik 118

8.4 Feinschliff 126

8.5 Anmerkungen 127

9 Status und Ausblick 129

9.1 Status der Werkzeugunterstützung 129

9.2 Nächste Schritte 130

9.3 Ausdehnung auf andere Netzwerkkomponenten 130

Anhang A — Schnittstelle zur Befunddatenbank 131

Quellenverzeichnis 137

Liste der Abkürzungen 139

Copyright © Fraunhofer IESE 2003

Copyright © Fraunhofer IESE 2003

1 Problemstellung und Lösungsansatz

IP-basierte Kommunikationsnetze haben in den vergangenen Jahren an Bedeutung gewonnen. Immer mehr Unternehmen sind in an das Internet angeschlossen. Dies setzt die unternehmensinterne Informationstechnologie (IT) der Gefahr sogenannter Cyber-Attacken aus.

1.1 Netzsicherheit durch sichere Router-Konfiguration

Wesentlichen Einfluß auf die Sicherheit von IP-Netzen haben die Router, mit deren Hilfe IP-Pakete zwischen Sendern und Empfängern vermittelt werden. Moderne Router übernehmen nicht nur Vermittlungsfunktionen, sondern dienen zugleich auch der Filterung, Separierung und Überwachung von Datenströmen. Darüber hinaus beherbergt das Router-Betriebssystem oft zahlreiche Standarddienste — zum Beispiel Telnet, TFTP oder SNMP — von denen leicht Gefahren für die Netzkomponenten ausgehen können.

Router müssen mit großer Sorgfalt konfiguriert werden, um Bedrohungen durch Ausspähen, Stören, Verfälschen oder Umlenken von IP-Verkehr abzuwehren und sicherzustellen, daß der Router Angreifern keinen Stützpunkt für weitergehende Angriffe auf das Netz bietet. Das Konfigurieren eines Routers ist jedoch eine schwierige Aufgabe.

1.2 Werkzeugunterstützung für Router-Sicherheitsrevisionen

Routerkonfigurationen sind unter Umständen sehr komplex und lassen sich nicht mehr ohne weiteres überblicken. Entsprechend hoch ist das Risiko des Netzwerkadministrators, Konfigurationsfehler zu begehen, die bei manueller Prüfung unentdeckt bleiben. Hier kann geeignete Werkzeugunterstützung erheblich dazu beitragen, verborgene Schwachstellen in der Konfiguration eines Routers aufzuspüren.

CROCODILE ist ein solches Prüfwerkzeug. Es liest Konfigurationsbeschreibungen im Textformat ein und durchsucht diese nach potentiellen Konfigurationsmängeln. Die Analyse basiert auf Prüfkriterien, die teils fest vorgegeben, teils frei konfigurierbar sind. Der Prüfbefund wird als Hypertext-Dokument ausgegeben.

Copyright © Fraunhofer IESE 2003

Das Prüfwerkzeug ist derzeit für Router des Herstellers Cisco Systems ausgelegt, die unter dem Betriebssystem IOS betrieben werden. Diese Routerklasse ist derzeit marktbeherrschend. Zwar beziehen sich alle Prüfkriterien auf IOS-Routerkonfigurationen. Jedoch ist das Prüfwerkzeug als modulares, IOS-unabhängiges Framework konzipiert. Grundsätzlich ist es daher möglich, durch Austausch der Prüfmodule auch andere Gerätetypen — zum Beispiel Cisco PIX Firewalls — in gleicher Weise zu überprüfen. Die Voraussetzung für den Einsatz des CROCODILE Frameworks besteht lediglich darin, daß die Konfigurationsbeschreibung im Textformat — als Folge einzelner Konfigurationsklauseln — vorliegt.[2]

Bei der Konzeption der Werkzeugunterstützung standen folgende Anforderungen im Vordergrund:

- **Automatisierung**
  Das Werkzeug soll einfach anzuwenden sein und den manuellen Spezifikations- und Prüfaufwand minimieren.

- **Konfigurierbarkeit**
  Das Werkzeug soll möglichst einfach an wechselnde Sicherheitsanforderungen und Prüfgegenstände anpaßbar sein.

- **Portabilität**
  Das Werkzeug soll auf unterschiedlichsten Plattformen ablauffähig sein, insbesondere auf gängigen Unix- oder Windows-Systemen. Installation und Deinstallation sollen problemlos möglich sein.

- **Modularität**
  Das Werkzeug soll möglichst einfach zu erweitern sein — auch durch den Anwender selbst.

In den nachfolgenden Kapiteln wird erläutert, wie sich diese Anforderungen im Entwurf des Prüfwerkzeugs widerspiegeln.

1.3 Sicherheitsprüfkriterien

Den Nutzen des Prüfwerkzeugs bestimmt vor allem die Güte seiner Prüfkriterien. Die vom Werkzeug verwendeten Prüfkriterien leiten sich aus öffentlich verfügbaren Empfehlungen zur sicheren Konfiguration von Routern ab.

2 Probleme würden hingegen Netzwerkkomponenten bereiten, die ausschließlich über eine graphische Benutzerschnittstelle konfigurierbar sind und deren Konfigurationseinstellungen sich nicht im Textformat exportieren lassen. Einige Routerhersteller speichern Konfigurationsattribute ausschließlich in einem proprietären Binärformat. Hier wäre der CROCODILE-Ansatz ungeeignet.

Copyright © Fraunhofer IESE 2003

Die wichtigsten uns bekannten Quellen für solche Sicherheitsempfehlungen liefert der Hersteller Cisco. Dazu zählen verschiedene Bände der Cisco IOS Reference Library, insbesondere [IOS12.0a] und [IOS12.0b]. Außerdem hält Cisco im Internet verschiedene Publikationen zum Download bereit, zum Beispiel [CiscoTech]. Für Netzbetreiber ist die Cisco-Empfehlung «Essential IOS Features Every ISP Should Consider» [IOSessentials] der entscheidende Konfigurationsmaßstab. Die «Essentials» werden häufig aktualisiert und repräsentieren derzeit wohl am besten, was in diesem Bereich als gute Praxis gilt.

Sehr detaillierte Informationen bietet auch der aktuelle «Router Security Configuration Guide» der U.S. National Security Agency (NSA) [NSAguide]. Er enthält konkrete Konfigurationsanweisungen für IOS-Router. Inhaltlich decken sich [NSAguide] und [IOSessentials] in weiten Teilen, wobei der NSA-Leitfaden die notwendigen Maßnahmen oft etwas anschaulicher und präziser beschreibt. Bezüglich einiger Konfigurationsempfehlungen sind die beiden Quellen allerdings widersprüchlich. Dies ist darauf zurückzuführen, daß die Anforderungen von Netzbetreibern nur schwer mit den Bedürfnissen der Betreiber lokaler, privater Netze vereinbar sind. Sicherheitsanforderungen müssen immer im Kontext des konkreten Anwendungsfalls gesehen werden!

Im folgenden verzichten wir darauf, inhaltlich näher auf die verwendeten Prüfkriterien einzugehen. In Kapitel 4 skizzieren wir die verfügbaren Prüfmodule und deren Prüfumfang in groben Zügen, ohne die Bedeutung einzelner Prüfschritte im einzelnen zu erläutern. Diesbezüglich verweisen wir auf die vorstehend genannten Quellen.

1.4 Gliederung des Handbuchs

Das vorliegende Handbuch beschreibt Eigenschaften, Architektur und Funktionsweise des Prüfwerkzeugs. Kapitel 2 zeigt im Überblick, wie das Prüfwerkzeug eingesetzt wird und welche Systemvoraussetzungen dazu notwendig sind. Kapitel 3 beschreibt den Aufbau und die Entwurfsphilosophie des Werkzeugs. Es skizziert das Zusammenspiel der einzelnen Werkzeugkomponenten.

Kapitel 4 ist den verfügbaren Prüfmodulen gewidmet. Es beschreibt deren Funktionsweise und Prüfumfang. Kapitel 5 stellt ein spezielles Prüfmodul vor, das die Funktionsweise des Router Audit Tools (RAT) — einem im Internet frei verfügbaren IOS Checker Tool — exakt nachbildet, um RAT-Anwendern den Umstieg auf CROCODILE zu erleichtern.

Kapitel 6 ist dem Stapelverarbeitungsbetrieb gewidmet. Es stellt dar, wie der Anwender mit CROCODILE Reihenuntersuchungen an vielen IOS-Konfigurationsdateien durchführen kann, und wie sich dabei maschinenlesbare Befunddaten für eine Nachbearbeitung mit unabhängigen Werkzeugen erzeugen lassen.

Copyright © Fraunhofer IESE 2003

In Kapitel 7 werden verschiedene Hilfswerkzeuge vorgestellt, die das Werkzeug ergänzen und abrunden.

Kapitel 8 wendet sich an Entwickler, die das Werkzeug um weitere Prüfmodule erweitern wollen. Es vermittelt einen groben Überblick über die Vorgehensweise bei der Erstellung eines Moduls.

Abschließend bietet Kapitel 9 eine kurze Zusammenfassung und einen Ausblick auf künftige Weiterentwicklungen.

Copyright © Fraunhofer IESE 2003

2 Anwendung des Prüfwerkzeugs im Überblick

Bevor wir den Anwender in einer länglichen Darstellung mit den vielschichtigen Eigenschaften des Werkzeugs vertraut machen, wollen wir im folgenden kurz die wesentlichen Schritte des Werkzeugeinsatzes zusammenfassen. Diese Kurzeinführung sollte den Anwender in die Lage versetzen, das Program zu installieren und einen einfachen Standardprüflauf durchzuführen.

2.1 Systemvoraussetzungen

Hardware und Betriebssystem

Das Werkzeug benötigt keine besondere Hardware. Ein handelsüblicher Intel-PC mit mindestens 100 MHz Taktfrequenz, 128 MB Hauptspeicher und einigen Megabyte freiem Plattenplatz, der unter Linux oder Windows betrieben wird, genügt den Anforderungen.

Das Werkzeug ist keineswegs auf Intel-Plattformen festgelegt, sondern läuft problemlos auch auf anderen Architekturen. Entwickelt und getestet wurde CROCODILE größtenteils unter Sun Solaris auf einem SPARC-Prozessor.

PERL Laufzeitumgebung

Da das Werkzeug in PERL implementiert ist, benötigt es zur Ausführung eine PERL-Laufzeitumgebung. PERL gehört bei gängigen Unix- bzw. Linux-Distributionen zum Standardumfang. Für Windows-basierte Plattformen sind vorgefertigte PERL-Executables oder wahlweise Quellcode-Archive im Internet frei verfügbar.

Achtung: *Je nach verwendetem Betriebsystem und installierter PERL-Distribution kann die Ausführungsgeschwindigkeit erheblich variieren. Bei einem unserer Tests benötigte die Analyse eines Konfigurationsbeispiels unter Windows die 14-fache Laufzeit eines entsprechenden Laufs unter Linux auf dem gleichen Rechner. Da PERL ursprünglich für Unix konzipiert wurde, ist davon auszugehen, daß Unix- bzw. Linux-Distributionen von PERL im Zweifelsfalle performanter als vergleichbare Windows-Distributionen sind.*

Copyright © Fraunhofer IESE 2003

Die vorliegende Software wurde unter den PERL Versionen 5.4, 5.6 und 5.8 entwickelt und unter den Betriebssystemen Sun Solaris 2.6 und 2.8, RedHat Linux Kernel 2.4.9 sowie Windows 2000 erfolgreich getestet. Um die Portabilität des Codes zu gewährleisten, haben wir bewußt darauf verzichtet, PERL-Sprachmittel neuerer Versionen als Version 5.4 einzusetzen. Auch haben wir vermieden, PERL-Zusatzmodule zu verwenden, die über eine Standarddistribution hinausgehen.

2.2 Installation und Deinstallation des Prüfwerkzeugs

Wir setzen voraus, daß PERL auf der Ausführungsplattform bereits installiert worden ist. Für das Werkzeug selbst bedarf es dann keiner besonderen Vorbereitungen mehr. Auf einem Unix-System muß der Anwender zum Beispiel die folgenden Schritte ausführen, um CROCODILE zu installieren:

1. Einloggen und Wechsel in ein freies Unterverzeichnis, zum Beispiel in das Verzeichnis '/tmp'

```
user@host[~]        cd /tmp
user@host[/tmp]
```

2. Laden des Werkzeugarchivs und Entpacken der Software

```
user@host[/tmp]     cp /floppy/crocodile.tar  .
user@host[/tmp]     tar xvf crocodile.tar
```

3. Wechsel in das Werkzeug-Unterverzeichnis

```
user@host[/tmp]     cd CROCODILE
```

Damit ist das Werkzeug einsatzbereit. Um die Installation wieder zu entfernen genügt es, das Unterverzeichnis 'CROCODILE' — oder wie immer der Installationspfad lautet — zu entfernen. CROCODILE greift nicht auf Daten außerhalb seines Installionsverzeichnisses zu, sofern der Anwender nicht ausdrücklich andere Eingabe- oder Ausgabe-Pfade spezifiziert.

2.3 Auswahl der Eingabedaten

CROCODILE erwartet als Eingabe eine IOS Routerkonfiguration in gewöhnlichem Textformat, so wie sie zum Beispiel das IOS-Kommando 'show running-config' liefert. Die «running config» ist ideal, denn

- sie zeigt immer den wahren Konfigurationszustand des Routers im laufenden Betrieb;

Copyright © Fraunhofer IESE 2003

- alle IOS-Schlüsselwörter sind in voller Länge ausgeschrieben (was bei manueller Konfiguration nicht erforderlich ist).

Ein gewisser Nachteil der «running config» besteht darin, daß Voreinstellungen (defaults) meist nicht explizit ausgewiesen sind.

Es spricht auch nichts dagegen, eine handgeschriebene IOS-Konfiguration in CROCODILE einzuspeisen, denn das Werkzeug setzt weder die spezifische Ausgabereihenfolge von 'show running-config' noch dessen charakteristische, Konfigurationsmodus-abhängige Einrückung des Konfigurationstexts voraus. Verwendet der Nutzer als Datenquelle eine extern hinterlegte Konfigurationsbeschreibung (z.B. von einem TFTP-Server stammend), so muß er darauf achten, daß im Konfigurationstext alle Schlüsselwörter ausgeschrieben sind, denn CROCODILE ignoriert — von wenigen Ausnahmen abgesehen — Kurzformen. Außerdem muß der Anwender sicherstellen, daß die untersuchte Konfigurationsbeschreibung mit der in Betrieb befindlichen Konfiguration übereinstimmt.

Wir empfehlen daher, auf alle Fälle noch einmal die «running config» des fertig konfigurierten Routers auszulesen und zur Sicherheit noch einmal einem Analyselauf zu unterziehen.[3]

2.4 Probelauf

Im Werkzeugverzeichnis findet sich das Hauptprogramm 'crocodile' sowie verschiedene Unterverzeichnisse gemäß Abbildung 10, Seite 43. Im Unterverzeichnis 'SampleConfigs' sind zu Demonstrationszwecken Beispielkonfigurationen hinterlegt. Eine Analyse einer Konfigurationsbeschreibung erhält man, indem man den Inhalt der Konfigurationsdatei zum Beispiel[4] wie folgt in das PERL-Programm einspeist:

```
user@host[CROCODILE]  perl crocodile SampleConfigs/acsac.cfg
```

Der Aufruf leitet die Datei 'acsac.cfg' als Eingabe zum PERL-Programm 'crocodile'.[5] Status- und Fehlermeldungen werden auf der Konsole protokolliert.

---

3 Außerdem ist zu prüfen, mit welcher Konfiguration der Router nach einem Reboot hochfährt: Handelt es sich dabei tatsächlcih um die aktuelle «running config»?

4 Der Aufruf 'perl crocodile <*Eingabedatei*>' ist die kanonische Form, die plattformunabhängig stets funktionieren sollte. Je nach verwendetem Betriebssystem kann der Benutzer das Kommando zu 'crocodile <*Eingabedatei*>' verkürzen, sofern er die Programmdatei 'crocodile' mit dem PERL-Programmpfad seines Systems verknüpft: Sie dazu die Kopfzeile des 'crocodile' Hauptprogramms!

5 Das Programm ist so ausgelegt, daß es wahlweise auch die Standardeingabe liest. Unter Unix zum Beispiel sind daher auch die folgenden Aufrufe möglich: (1) Piping innerhalb einer Prozeßkette — 'cat <*Eingabedatei*> | crocodile', (2) Eingabeumlenkung — 'crocodile < <*Eingabedatei*>'

Copyright © Fraunhofer IESE 2003

Im Beispiel wird vorausgesetzt, daß der PERL-Interpreter im Suchpfad des Benutzers liegt. Andernfalls muß statt 'perl' der absolute Pfadname angegeben werden, unter dem PERL installiert ist — zum Beispiel '/usr/local/bin/perl'. Nach dem Programmstart erscheint auf der Konsole in etwa die folgende Ausgabe:

```
CROCODILE started.

Evaluation Target:  SampleConfigs/acsac.cfg

Reading '/tmp/CROCODILE/Configuration/plugin.conf' ...
... done (plugin.conf)

Reading '/tmp/CROCODILE/Configuration/linkdata.conf' ...
... done (linkdata.conf)

Reading '/tmp/CROCODILE/Configuration/designators.conf' ...
... done (designators.conf)

Reading '/tmp/CROCODILE/Configuration/IngressEgress.conf' ...
... done (IngressEgress.conf)

Reading '/tmp/CROCODILE/Configuration/Passwords.conf' ...
... done (Passwords.conf)

Reading '/tmp/CROCODILE/Configuration/CompoundPatterns.conf' ...
... done (CompoundPatterns.conf)

Passwords PLUGGED IN.

CompoundPatterns PLUGGED IN.

Connectivity PLUGGED IN.

NTP PLUGGED IN.

IngressEgress PLUGGED IN.

AAA PLUGGED IN.

SNMP PLUGGED IN.

Logging PLUGGED IN.

Starting parse ...
    114 lines of configuration parsed.

Parsing finished. Now analyzing ...

Checking for trivial No. 5 passwords ...

Dumping subnet information to result database ...
```


Copyright © Fraunhofer IESE 2003

```
Dumping node information to result database ...

Writing interface info to '/tmp/CROCODILE/XML_data' ...

Writing line interface info to '/tmp/CROCODILE/XML_data' ...

Writing access-list info to '/tmp/CROCODILE/XML_data' ...
  Skipping 'ACL 110' (empty!): Used but not defined?

Determine whether interfaces accept  NTP requests:
  Serial0
  Ethernet0
  -- undefined ACL '110' --

Ingress Restrictions for Interfaces ...

Egress Restrictions for Interfaces ...

Ingress Restrictions for Lines ...

Egress Restrictions for Lines ...

Restrictions for Accesslists ...

Reading '/tmp/CROCODILE/Output/HTML_output.conf' ...
... done (HTML_output.conf)

Writing Results ...

Analysis finished. See '/tmp/CROCODILE/HTML_results/index.html'.

Successful termination: No PROBLEMS and no ERRORS.
user time 0:19 min   system time 0:00 min
```

Die Analyse von 'acsac.cfg' benötigt — je nach dem Leistungsvermögen des Rechners — nur wenige Sekunden. Um den Fortschritt der Berechnungen anzuzeigen, erscheint in der Konsolenausgabe mitunter eine Zeilen- oder Prozentangabe, die mit dem Fortschritt der Auswertung hochgezählt wird.

2.5 Darstellung der Prüfergebnisse

Die Ergebnisse eines Prüflaufs werden als Hypertext im HTML-Format in verschiedenen Ansichten aufbereitet und im Unterverzeichnis 'HTML_results' unter der Einstiegsseite 'index.html' abgelegt. Zur Darstellung genügt ein herkömmlicher HTML-Browser, zum Beispiel Netscape Navigator, Mozilla, Internet Explorer oder Konqueror (Abbildung 1).

Das Werkzeug nutzt zur Strukturierung der Hypertext-Dokumente zum einen HTML Frames, zum anderen ein zentrales Cascading Style Sheet, zu finden unter

Copyright © Fraunhofer IESE 2003

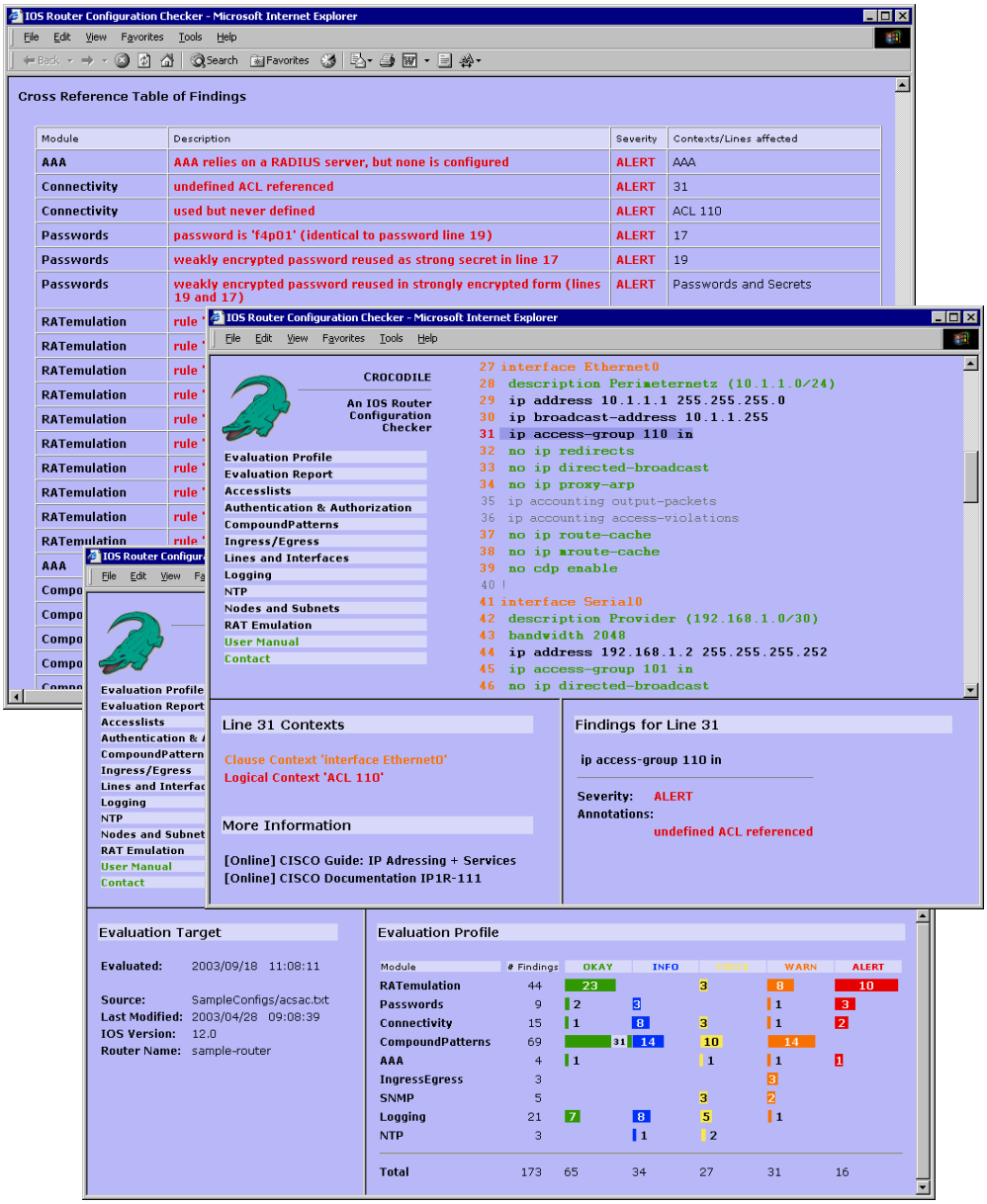

Abbildung 1 Prüfergebnisse als HTML-Hypertext

'HTML_results/default.css'. Der Rückgriff auf diese Formatierungshilfsmittel ermöglicht eine übersichtliche, flexibel anpaßbare Dokumentenstruktur. Allerdings muß der verwendete Browser geeignet konfiguriert sein, damit er statt seiner eigenen Default-Einstellungen die Style-Sheet–Vorgaben verwendet. Moderne Browser sollten sowohl Frames als auch Style Sheets problemlos unterstützen. Welche Einstellungen dazu im einzelnen erforderlich sind, ist jedoch Browser-spezifisch.

Copyright © Fraunhofer IESE 2003

Achtung: *Bei falscher Browser-Einstellung ergibt sich ein unbefriedigendes, in der Regel jedoch noch verwendbares Ausgabeformat. Um wohlproportionierte Fenstergrößen, gut erkennbare Farbzuordnungen und ansprechende Schriftgrößen zu erhalten, muß der Anwender die Browser-Einstellungen geeignet wählen, damit das vorgesehene Style Sheet korrekt ausgewertet wird. Beim Microsoft Internet Explorer 6.0 finden sich entsprechende Optionen zum Beispiel unter 'Tools → Internet Options …' und dort unter 'General → Accessibility'.*

Wichtige Parameter des HTML-Formats — zum Beispiel Schriftgrößen und Farbzuordnungen — können in der Datei 'Output/HTML_output.conf' den eigenen Wünschen angepaßt werden. Dies ist sinnvoll, denn unterschiedliche Browser interpretieren Farbcodes oder Angaben zur Schriftgröße zum Teil recht unterschiedlich. Der Anwender kann die Darstellungsparameter entsprechend der verwendeten Systemplattform geeignet nachjustieren und bei Bedarf ein bereits vorhandenes Analyseergebnis mit Hilfe der Utility 'Utils/make_html.pl' neu aufbereiten, ohne dazu den Prüflauf wiederholen zu müssen (siehe dazu Kapitel 7.2).

2.6 Laufzeitoptionen und Help-Funktion

Die Ausführung des Hauptprogramms 'crocodile' kann durch verschiedene Aufrufparameter beeinflußt werden. So ist es zum Beispiel möglich, den Detaillierungsgrad der Konsolenmeldungen ("verbosity") zu Debug-Zwecken stufenweise zu erhöhen oder alle Prüflauf-Statusmeldungen in eine Datei zu protokollieren ("Logging").

Die Beschreibung aller zulässigen Laufzeit-Optionen würde an dieser Stelle zu weit führen. Für eine erste Inbetriebnahme sind voraussichtlich keine besonderen Kommandozeilenparameter erforderlich.

CROCODILE ist weitgehend selbsterklärend. Es besitzt eine Online-Hilfefunktion (Option '-h') zur schnellen Orientierung. Eine Kurzbeschreibung der Laufzeit-Optionen erhält der Anwender zum Beispiel mit folgendem Aufruf:

```
user@host[CROCODILE]  perl crocodile -h
USAGE

  crocodile [ <options> ] [ <IOS config file> ]


DESCRIPTION

  CROCODILE reads an IOS router configuration file in ASCII format,
  analyzes the configuration settings, and creates evaluation reports
  in ASCII, HTML, or XML format, depending on command line options.
  If no file name (or filename '-') is specified, STDIN is read.
```

Copyright © Fraunhofer IESE 2003

```
OPTIONS

  The options below are general option for both interactive and
  batch mode.

 -c <dir> Configuration directory where CROCODILE *.conf files can be
          found; default is './Configuration'

 -h       Show this help information on console and exit

 … weitere Optionen folgen …
```

Etliche der Optionen werden hauptsächlich intern verwendet, um 'crocodile' im Stapelverarbeitungsmodus zu betreiben (siehe Kapitel 2.7). Im interaktiven Betrieb spielen die meisten Laufzeitparameter nur eine untergeordnete Rolle.

2.7 Die nächsten Schritte …

Nachdem sich der Anwender grundsätzlich mit 'crocodile' vertraut gemacht hat, kann er gezielt Prüfmodule auswählen, um damit Routerkonfigurationen zu analysieren. Er kann auch eigene Prüfregeln formulieren. Das genaue Vorgehen beschreibt Kapitel 4.

Neben der interaktiven Analyse einzelner Konfigurationsdateien ermöglicht CROCODILE auch die Überprüfung vieler Konfigurationen im Stapelverarbeitungsmodus (batch mode). Dazu dient ein separates Frontend: 'batchcroco'. Dieses Programm verfügt ebenfalls über eine eingebaute Online-Hilfe mit Kurzerklärung aller Aufrufparameter (Option '-h'):

```
user@host[CROCODILE]  perl batchcroco -h
USAGE

  batchcroco [ <options> ] <IOS config files or directories> ...

DESCRIPTION

  This is the batch processing frontend for CROCODILE.  It reads in
  a set of IOS router configuration files in text format, and feeds
  them into CROCODILE for analysis. The results of all evaluations are
  collected. See './crocodile -h' for more details.


OPTIONS

  -d <dir>        Store output in <dir> (default: current directory)

  -f <fmt>        Generate output format <fmt>: ASCII or HTML

  … weitere Optionen folgen …
```


Copyright © Fraunhofer IESE 2003

Eine genaue Beschreibung des Stapelverarbeitungs-Frontends findet sich in Kapitel 6. Alle Kommandozeilen-Optionen lassen sich auch mittels der Konfigurationsdatei './Configuration/batch.conf' vorbelegen; die Konfigurationsdatei enthält genauere Erläuterungen zu Bedeutung und Format der einzelnen Konfigurationsparameter.

Copyright © Fraunhofer IESE 2003

Copyright © Fraunhofer IESE 2003

3 Architektur und Funktionsprinzip

Das folgende Kapitel beschreibt Aufbau und Wirkungsweise des Prüfwerkzeugs. Der Entwurf ist auf größtmögliche Flexibilität ausgelegt, um das Werkzeug mit wenig Aufwand auf neue oder geänderte Prüfanforderungen umrüsten zu können.

3.1 Übersicht

Werkzeugkomponenten

Das Werkzeug gliedert sich grob in drei Komponententypen: ein Mustererkennungsmodul (Parser), eine beliebige Anzahl von Prüfmodulen sowie ein Darstellungsmodul zur Ergebnisaufbereitung. Jedes Prüfmodul realisiert eine in sich abgeschlossene Prüfaufgabe mit allen logischen Prüf- und Interpretationsschritten, die erforderlich sind, um zu einem Prüfbefund zu gelangen. Der Parser bildet das Bindeglied zwischen den Prüfmodulen und den Eingabedaten. Die Darstellungskomponente sorgt für eine übersichtliche graphische Präsentation des Untersuchungsbefunds.

Arbeitsweise

Das Prüfwerkzeug arbeitet nach folgendem Schema. Der Parser liest eine gültige Cisco-Konfigurationsdatei im Textformat ein. Er vergleicht Textzeile für Textzeile mit einer Menge vorgegebener Textmuster. Sobald ein Textmuster (Pattern) im Konfigurationstext erkannt wird, benachrichtigt der Parser alle Prüfmodule, die sich für das betreffende Muster interessieren, und übergibt ihnen die dem Muster entsprechende Textzeile zur Auswertung.

Erhält ein Prüfmodul eine Benachrichtigung, so führt es eine dem Textmuster zugeordnete Prüfroutine (Pattern Handler) aus. Der Pattern Handler ist eine Methode des Prüfmoduls und umfaßt alle logischen Prüfschritte, die beim Auftreten des Textmusters vorgesehen sind. Der Methodenaufruf liefert ein Prüfergebnis sowie gegebenenfalls ergänzende Kommentare zum Prüfergebnis. Alle Befunde werden in strukturierter Form in eine interne Ergebnisdatenbank eingespeist.

Copyright © Fraunhofer IESE 2003

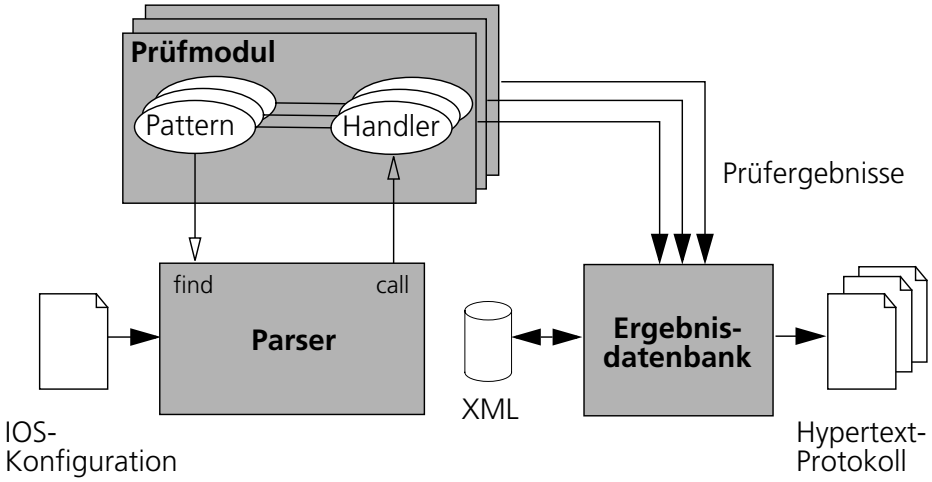

Abbildung 2 Übersicht über Struktur und Funktionsweise des Prüfwerkzeugs

Sind alle Textzeilen der Eingabedatei in dieser Weise verarbeitet worden, so wird reihum jedes Prüfmodul per Methodenaufruf noch einmal aufgefordert, eine abschließende Gesamtwertung seines jeweiligen Prüfbefunds zurückzumelden. Dies bietet den Modulen die Gelegenheit, aus der Summe aller ihnen gemeldeten Textzeilen eine integrierte Sicht auf den zu prüfenden Sicherheitsaspekt zu konstruieren, die sich nicht an einer einzelnen Textzeile festmachen läßt.

Die Rückmeldungen aller Module ergänzen die in der Datenbank gesammelten Befunde. Nach Abschluß des Parser-Laufs wird die gesamte Befunddatenbank als Datei im XML-Format gespeichert. Diese Rohdaten werden genutzt, um daraus verschiedene HTML-Sichten zu generieren, die sich mit einem gewöhnlichen Web-Browser darstellen lassen. Abbildung 2 zeigt das Funktionsprinzip in der Übersicht.

Prüfmodule als «Software-Einschübe»

Das Prüfwerkzeug wird für einen Prüflauf vorbereitet, indem der Anwender die vorgesehenen Prüfmodule in den Parser «einsteckt». Alle Prüfmodule verfügen über eine genormte Schnittstelle, die genau zu einer entsprechenden Andockmöglichkeit des Parsers paßt. Dem Grundsatz nach sind Prüfmodule vollkommen unabhängig von einander. Sie können einzeln oder auch in Gruppen in das Parser-Modul eingesteckt werden, um einen Prüflauf mit individuellem Prüfumfang zusammenzustellen.

Dem Anwender steht es frei, für jeden zu prüfenden Sicherheitsaspekt ein separates Prüfmodul bereitzustellen oder logisch verwandte Prüfpunkte in einem

Copyright © Fraunhofer IESE 2003

gemeinsamen Modul zusammenzufassen. Im Extremfall ist für jedes einzelne Textmuster ein eigenes Prüfmodul zuständig. Meist übernimmt jedoch ein Prüfmodul die Analyse mehrerer, mitunter dutzender von Textmustern, die sich alle auf einen übergeordneten logischen Sicherheitsaspekt beziehen. Es spricht auch nichts dagegen, das gleiche Textmuster von mehreren Prüfmodulen — jeweils unter einem anderen Gesichtspunkt — analysieren zu lassen. Die Wahl der Modulgranularität sollte sich ausschließlich nach den Bedürfnissen des Anwenders richten. Maßgeblich für den Modulumfang sind vor allem die Übersichtlichkeit und Bequemlichkeit der Programmierung sowie die Einfachheit der Werkzeuganwendung.

Natürlich können Prüfmodule bei Bedarf auch untereinander kooperieren, indem sie etwa gemeinsam genutzte Befund-Daten aufbauen und pflegen. In diesem Fall ist aber zu berücksichtigen, daß das einzelne Modul nicht mehr unabhängig von anderen Modulen eingesetzt werden kann, mit denen es kooperiert. Das Prüfergebnis ist nur dann vollständig und korrekt, wenn alle kooperierenden Module ebenfalls am Prüflauf beteiligt waren und ihren jeweiligen Befund beigetragen haben. Aus diesem Grunde empfiehlt es sich, bei der Konzeption von Prüfmodulen die Abhängigkeit von anderen Prüfmodulen möglichst zu minimieren. Jedes Prüfmodul hat die Möglichkeit, eine Liste aller Prüfmodule, von denen es abhängt («REQUIRED_MODULES»), zu deklarieren. Das Prüfwerkzeug lädt bei Programmausführung alle REQUIRED_MODULES automatisch nach, sofern diese versehentlich nicht in den Parser «eingesteckt» worden sind.

Programmiertechnische Realisierung

Das Prüfwerkzeug wurde in der Programmiersprache PERL (Practical Extraction and Report Language) implementiert. Für die Wahl dieser Sprache gab es mehrere Gründe:

- PERL ist frei verfügbar und auf unterschiedlichsten Hardware-Plattformen unter verschiedenen Betriebssystemen lauffähig. Insbesondere gibt es PERL-Laufzeitumgebungen für die verbreiteten Linux- und Windows-Systeme, was eine problemlose Portierung des Werkzeugs ermöglicht.

- PERL verfügt über sehr mächtige, zugleich sehr effiziente Operationen zur Textmanipulation, Mustersuche und dergleichen. Damit lassen sich Parser- und Analyse-Routinen elegant und kompakt formulieren, ohne daß darunter die Ausführungsgeschwindigkeit leidet.

- PERL ist objektorientiert. Komponenten eines bestimmten Komponententyps können daher gemeinsame Eigenschaften mittels Vererbung von einem Basistyp erwerben. So erben etwa alle Prüfmodule ihre Andock-

Copyright © Fraunhofer IESE 2003

Schnittstelle zum Parser von der gemeinsamen Basisklasse 'Module'; bei der Erstellung neuer Prüfmodule muß dafür kein gesonderter Programmcode mehr geschrieben werden, was den Programmierer deutlich entlastet.

- PERL-Programme werden normalerweise von einem Interpreter ausgeführt, bedürfen also keiner Übersetzung und Bindung (Compiling, Linking) vor ihrem Start. Dies begünstigt Rapid Prototyping bei der Erstellung neuer Prüfmodule.

Die Werkzeugkomponenten werden mit Hilfe eines sehr einfachen Hauptprogramms zu dem Gesamtwerkzeug integriert (Abbildung 3). Das Hauptprogram umfaßt in seinem Kern folgende Abschnitte:

1. Zunächst werden alle benötigten Werkzeugkomponenten deklariert. Sofern der Anwender eigene Prüfmodule ergänzen möchte, sind diese mittels 'use'-Klauseln bekanntzugeben.

2. Als nächstes sind die Dateinamen für Ein- und Ausgabe zu bestimmen. (Abweichend von Abbildung 3 wird man hier üblicherweise Aufrufparameter auswerten. Im Beispiel oben wurde der Übersichtlichkeit halber darauf verzichtet.) Für die Generierung der verschiedenen XML- und HTML-Ausgabeformate werden komfortable PERL-Objekte mit geeigneten Unterstützungsfunktionen verwendet.

3. Anschließend wird je eine Instanz des Parsers und aller gewünschten Prüfmodule erzeugt. Die Prüfmodule werden sodann mittels 'plugin()' in den Parser eingesteckt.

4. Damit ist der Parser startbereit. Der Text wird zeilenweise einem Parsing unterzogen. Daraus resultierende Befunde werden von den jeweiligen Prüfmodulen in die Ergebnisdatenbank geschrieben.

5. Nach Abschluß des Parsens führt der Parser mit allen angeschlossenen Modulen eine Abschlußanalyse durch, bei der jedes Modul aus einer spezifischen Analysesicht sein Gesamtfazit zieht.

6. Das Programm endet mit dem Ausprotokollieren der Analysedaten in verschiedenen Text-, XML- und HTML-Formaten.

Wie man sieht, ermöglicht die Programmstruktur auf sehr einfache Weise, das Werkzeug um neue Module zu ergänzen. Indem der Anwender seine neu erstellten Prüfmodule von der PERL-Klasse 'Module' ableitet ist sichergestellt, daß diese sich mittels 'plugin()' problemlos einfügen lassen.

Copyright © Fraunhofer IESE 2003

```
# Deklaration der verwendeten Komponententypen:
use Parsing::Parser;
use Auxiliary::ResultData;
use Output::HTML_control;

use Plugin::CompoundPatterns;
use Plugin::IngressEgress;
# … nach Bedarf neue Plugin-Prüfmodule ergänzen

print "\nCROCODILE started ...\n\n";

# Name der verschiedenen Eingabe- und Ausgabedateien und -Pfade
my $input = "-";
my $html_outdir="./HTML_results";
my $xml_outdir = "./XML_data";
my $xml_output = "$xml_outdir/results.xml";

# Ergebnisdarstellung anlegen und initialisieren
my $html=HTML_control->new("Output/HTML_output.conf");

# Erzeugung und Initialisierung je einer Instanz der Prüfmodule
my (@modulelist, $module);
push @modulelist, Plugin::CompoundPatterns->new();
push @modulelist, Plugin::IngressEgress->new();
# … pro gewünschtem Plugin-Modul eine Instanz erzeugen und pushen

# Erzeuge und initialisiere den Parser: Checkermodule einstecken
my $parser = Parser->new($input);
foreach $module (@modulelist) {
   $parser->plugin($module);
   }

# Eingabedatei Zeile für Zeile parsen
print "\nStarting parse ...\n\n";
$parser->parse();

# Abschlußbewertung erstellen
print "\nParsing finished. Now analyzing ...\n\n";
$parser->analyze();

# Ergebnisdaten in den verschiedenen Formaten schreiben
print "\nWriting Results... \n";
my $data=ResultData->get();
$data->save_as_xml($xml_output);
$html->write($html_outdir);

print STDERR "\nAnalysis finished.\n\n";
```

Abbildung 3 Struktur des CROCODILE Hauptprogramms

Copyright © Fraunhofer IESE 2003

3.2 Parser-Modul

Ein Arbeitsschritt, der allen Prüfpunkten zugrundeliegt, besteht darin, in der Router-Konfigurationsbeschreibung vorgegebene Textmuster aufzuspüren und in ihre syntaktischen Bestandteile zu zerlegen. Das Parser-Modul übernimmt diese Aufgabe stellvertretend für alle Prüfmodule. Beim Einstecken eines Prüfmoduls mittels 'plugin()' übermittelt das Modul dem Parser eine Menge von Musterspezifikationen, die der Parser im Auftrag des Prüfmoduls detektieren soll. Tritt eines der vorgegebenen Muster auf, so wird das Prüfmodul verständigt und erhält die auf das Muster passende Textzeile — zerlegt in ihre Bestandteile gemäß Pattern-Spezifikation — zur Weiterverarbeitung.

Patterns

Relevante Textmuster — sogenannte Patterns — werden in Backus-Naur-Form beschrieben. Eine Pattern-Beschreibung setzt sich aus einem oder mehreren Grundbestandteilen (Atomen) zusammen, die zu Komposita kombiniert werden können. Die Komposition der Bestandteile ist nach folgenden Regeln möglich:

- **Sequenz**
  Die Aneinanderreihung '*<pattern1>* *<pattern2>* …' ergibt ein neues Pattern.

- **Auswahl**
  '{ *<pattern1>* | *<pattern2>* | … }' bedeutet: «Entweder *<pattern1>* oder *<pattern2>* oder … .»

- **Option**
  '[…]' bedeutet: «Der Inhalt der Klammer ist optional.»

- **Wiederholung**
  '{ … }*' bedeutet: «Der Inhalt der geschweiften Klammer ist wenigstens einmal vorhanden und wiederholt sich beliebig oft.»
  '[…]*' bedeutet: «Der Inhalt der eckigen Klammer fehlt oder wiederholt sich beliebig oft.»

- **Negation**
  '! *<pattern>*' bedeutet: «Das Pattern tritt (an dieser Stelle) nicht auf.»

Verfügbare Atoms

Zur Bildung von Patterns stehen eine Reihe vordefinierter Atome zur Verfügung. Bei Bedarf lassen sich leicht neue Atome im Parser nachrüsten. Tabelle 1 gibt einen Überblick über die derzeit definierten Atoms und ihre Bedeutung. Für jedes

Copyright © Fraunhofer IESE 2003

Atom ist, soweit möglich, eine äquivalente PERL Regular Expression angegeben, um formal zu definieren, welche Muster der Parser als gültigen Match akzeptiert.

Atom-Bezeichner	Regular Expression	Beschreibung
STRING	`^[^\s].*$`	Rest der Zeile ab dem ersten signifikanten Zeichen
WORD	`^[^\s!]+`	Zeichenkette ohne Whitespace (ohne angehängten Kommentar)
NAME	`^[A-Za-z][\w.-]*`	Wort aus Buchstaben, Punkten, Binde- und Unterstrichen
NUM	`^\d+`	Ganzzahl
IPADDR	`^\d+\.\d+\.\d+\.\d+`	IP-Adresse in Dot-Schreibweise, jede Zahl aus dem Bereich 0–255
DSTRINGD	`^D[^D]*D`	In Delimiterzeichen D eingeschlossene, ggf. mehrzeilige Zeichenkette
...	`^[^!]*`	Match mit dem verbleibenden Rest einer Zeile (ohne angehängten Kommentar)
PATTERN	–	Match mit einem gültigen BNF Pattern nach CROCODILE-Syntaxkonventionen
<Bereich> `NUM-NUM`	`^\d+`	Match mit einer Ganzzahl im angegebenen Intervall *<von>* – *<bis>*
<Numeral> `NUM['.' NUM]`	–	Match mit der gleichlautenden Ganz- oder Festkommanzahl
<Literal> *`'([^']*)'`*	–	Match mit gleichlautender, in Single Quotes eingeschlossener Zeichenfolge
<IOS Keyword> *`^[A-Za-z-][a-z-]*$`*	–	Match mit dem gleichlautenden IOS Schlüsselwort in beliebiger Groß/Kleinschreibung
<Makro> *`^[A-Z_][A-Z_\d]*$`*	–	Wird durch das zugeordnete CROCODILE Pattern substituiert (siehe Abschnitt «Makros», Seite 26)
<Regular Expression> *`\/(([^\/\\]|\\.)*)\/`*	–	Match mit gleichklautender PERL Regular Expression

Tabelle 1 Vordefinierte Atoms zur Bildung von CROCODILE Patterns

Eine typische Pattern-Spezifikation wäre zum Beispiel

```
access-list {1-99|1300-1999}{permit|deny}{any|IPADDR [IPMASK]} [log]
```

Copyright © Fraunhofer IESE 2003

In diesem Pattern sind 'access-list', 'permit', 'deny', 'any' und 'log' Schlüsselwörter der Cisco-Konfigurationssprache. 'IPADDR' bezeichnet das Atom für eine IP-Adressangabe, 'IPMASK' ist ein Makro, vordefiniert als ein Synonym für 'IPADDR'. Allgemein sind Atome (und Makros, siehe unten) an durchgehender Großschreibung erkennbar, während Schlüsselwörter in Kleinschrift, wahlweise mit führendem Großbuchstaben anzugeben sind. Die Bereichsangaben '1-99' und '1300-1999' stehen für eine beliebige Zahl aus dem angegebenen Bereich. Das Pattern oben würde zum Beispiel folgende Texte selektieren:

```
«access-list 3 permit any»
«access-list 1300 deny 143.69.0.0  0.0.255.255 log»
```

Die Pattern-Beschreibung wurde mit Bedacht so gewählt, daß sie der Darstellungsform einschlägiger Cisco-Handbücher entspricht. Der Anwender sollte wenig Mühe haben, zu einer vorgegebenen Konfigurationsanweisung das passende Pattern anzugeben.

Mit dem Atom-Typ «Regular Expression» besteht die Möglichkeit, native PERL-Ausdrücke innerhalb eines Patterns zu verwenden. PERL Regular Expressions ermöglichen mit wenig Aufwand einige sehr mächtige Konstrukte, zum Beispiel selektiert das Pattern

```
interface /^(?i)[a-z]+/ /^\d+(:\d+|\/\d+(\.\d+)?)*/ [...]
```

folgende Textzeilen:

```
«interface eth 0»
«interface ATM3/0.1 point-to-point»
«interface Serial2/1/7:20»
```

Allerdings sollten PERL-Ausdrücke nicht ohne Not verwendet werden, solange es passende CROCODILE-Atome gibt: Reguläre Ausdrücke sind nur schwer lesbar, und bei der Spezifikation unterlaufen dem Anwender leicht Fehler.

Pattern-Zerlegung

Wann immer ein vorgegebenes Pattern in der Eingabe des Prüfwerkzeugs erkannt wird, informiert der Parser alle betroffenen Prüfmodule und übermittelt ihnen die entsprechende Textpassage. Um den Prüfmodulen die Analyse möglichst zu erleichtern, liefert der Parser zu jeder gefundenen Textstelle die Daten in aufbereiteter Form. Die Angaben umfassen u.a.

- die **Zeilennummer**, in der die Textstelle in der Eingabe gefunden wurde;
- die **Textzeile** in ihrem vollen Wortlaut;

Copyright © Fraunhofer IESE 2003

- das **Pattern**, dem die Zeile entspricht;
- das jedem Pattern-Bestandteil entsprechende **Textfragment**.

Um jeder Pattern-Komponente (Atom oder Kompositum) das entsprechende Textfragment eindeutig zuzuordnen, werden die Komponenten hierarchisch numeriert. Im Beispiel oben ergäbe sich folgendes Numerierungsschema:

```
access-list {1-99|1300-1999}{permit|deny}{any|IPADDR [IPMASK]} [log]
  0:    access-list
  1:    {1-99|1300-1999}
  1.0:     1-99
  1.1:     1300-1999
  2:    {permit|deny}
  2.0:     permit
  2.1:     deny
  3:    {any|IPADDR [IPMASK]}
  3.0:     any
  3.1:     IPADDR
  3.2:     [IPMASK]
  3.2.0:       IPMASK
  4:    [log]
  4.0:     log
```

Das Prinzip der Nummernvergabe sollte offensichtlich sein. Die Pattern-Bestandteile werden von links nach rechts aufsteigend durchnumeriert, mit Null beginnend. Mit jeder öffnenden Klammer erfolgt ein Wechsel auf die nächsttiefere Numerierungsebene, wo das gleiche Schema rekursiv angewendet wird. Bei schließender Klammer erfolgt ein Aufstieg in der Numerierungsebene.

Wenn eine passende Textzeile zu diesem Pattern gefunden wird, dann ordnet der Parser jedem Pattern-Fragment das zugehörige Textfragment zu und meldet es unter der entsprechenden Fragmentnummer. Zum Beispiel erhält man für eine 'access-list'-Klausel nach obigem Syntaxformat folgende Zerlegung:

```
«access-list 1370 deny 143.69.0.0 0.0.255.255 log»
  0:     'access-list'
  1:     '1370'
  1.1:   '1370'
  2:     'deny'
  2.1:   'deny'
  3:     '143.69.0.0 0.0.255.255'
  3.1:   '143.69.0.0'
  3.2:   '0.0.255.255'
  3.2.0: '0.0.255.255'
  4:     'log'
  4.0:   'log'
```

Copyright © Fraunhofer IESE 2003

Die Analysemethode (Handler) des Prüfmoduls kann sich einfach auf die entsprechende Fragmentnummer des Patterns beziehen, um bestimmte Textfragmente anzusprechen. Die Numerierung erlaubt den geordneten Zugriff für jede beliebige Pattern-Spezifikation.

Konflikte: Gierige oder minmale Pattern-Interpretation?

Je nach gewähltem Pattern kann es mehrer Möglichkeiten geben, einen Match mit einem vorgegebenen Text zu bilden. Dies hängt damit zusammen, daß sich die Pattern-Bestandteile «gierig» oder «minimal» anwenden lassen.

Die «gierige» Strategie (Greedy Matching) versucht, jedem Atom einen möglichst langen Match zuzuordnen, ehe versucht wird, das nachfolgende Atom auf den verbleibenden Resttext anzuwenden. Die minimale Strategie versucht im Gegensatz dazu, jedem Atom nur einen kürzest möglichen Match zuzuordnen. Zwischen diesen beiden Extremen sind beliebige Mischstrategien denkbar.

Ein Beispiel soll das Problem verdeutlichen. Gegeben sei das Pattern

```
enable secret [level NUM] STRING
```

sowie der Text

```
«enable secret level 15 5 $1$iNeB$tr.JFRdI7ckzIX8v9g8c1».
```

Dann sind zwei Interpretationen des Patterns denkbar:

- Der optionale Pattern-Bestandteil '[level NUM]' ist anwendbar und kann daher einen Match mit dem Teiltext «level 15» erzeugen. Dem Atom 'STRING' ist dann als Match der Rest «5 1iNeB$tr.JFRdI7ckzIX8v9g8c1» zuzuordnen.

- Ein Match entsteht aber auch, wenn man den optionalen Pattern-Bestandteil '[level NUM]' überspringt. In diesem Falle wird STRING der Match «level 15 5 1iNeB$tr.JFRdI7ckzIX8v9g8c1» zugeordnet, dem optionalen Bestandteil '[level NUM]' hingegen der leere Match «».

Keine der beiden Strategien — weder die gierige noch die minimale — ist der anderen allgemein vorzuziehen, beide haben ihre Berechtigung. Für CROCODILE ist aber wichtig, daß ein Patterns immer eindeutig interpretierbar ist. Um dies zu gewährleisten, könnte man willkürlich eine Strategie zur Interpretation aller Patterns festlegen. Dies führt jedoch leicht zu Anwendungsfehlern, denn dem Benutzer fällt oftmals nicht auf, daß seine Pattern mehrdeutig sind, und oft hat er bei der Erstellung eines Patterns gerade nicht die gierige Interpretation im Sinne. Um Mißverständnissen vorzubeugen, löst CROCODILE das Problem auf

Copyright © Fraunhofer IESE 2003

andere Weise: Der CROCODILE-Parser prüft alle möglichen Interpretationen eines Patterns; wenn bei einer Klausel zwei unterschiedliche Interpretationen eines Patterns anwendbar sind, so meldet CROCODILE die Mehrdeutigkeit. Im Beispiel oben erscheint daher folgende Meldung:

```
*** Ambiguous pattern caused two conflicting pattern interpretations.
*** Pattern: 'enable secret [level NUM] STRING'
*** Text:   'enable secret level 15 5 $1$iNeB$tr.JFRdI7ckzIX8v9g8c1'
*** Match1:
***          0              'enable'
***          1              'secret'
***         3             'level 15 5 $1$iNeB$tr.JFRdI7ckzIX8v9g8c1'
*** Match2:
***          0              'enable'
***          1              'secret'
***          2              'level 15'
***          2.0            'level'
***          2.1            '15'
***          3              '5 $1$iNeB$tr.JFRdI7ckzIX8v9g8c1'
*** Conflicting attribute '3':
***    interpretation 1 is 'level 15 5 $1$iNeB$tr.JFRdI7ckzIX8v9g8c1'
***    interpretation 2 is '5 $1$iNeB$tr.JFRdI7ckzIX8v9g8c1'
```

Die Meldung besagt, daß der Pattern-Komponente Nummer 3 — also 'STRING' nach der kanonischen Zählung — bei Anwendung auf den angegebenen Text «enable secret level 15 5 1iNeB$tr.JFRdI7ckzIX8v9g8c1» zwei mögliche Matches zugeordnet werden können. Die beiden alternativen Deutungen werden ausgewiesen.

Damit der Prüflauf fehlerfrei durchlaufen kann, muß der Anwender das beanstandete Pattern so verfeinern, daß es eine eindeutige Interpretation hat. Im vorliegenden Falle wäre zum Beispiel folgende Umformulierung möglich:

```
enable secret [level NUM] !level STRING
```

In dieser Fassung darf der optionale Teil '[level NUM]' offensichtlich nur ausgelassen werden, wenn das Schlüsselwort 'level' im Text fehlt. Dies löst das Mehrdeutigkeitsproblem. Eine hundertprozentige, im vorliegenden Falle aber wohl unnötig aufwendige Lösungsvariante wäre

```
enable secret [level NUM] !{level NUM} STRING
```

Diese Version hat den Vorteil, daß sie auf schematische Weise ohne langes Nachdenken aus dem beanstandeten Pattern abgeleitet werden kann.[6]

6 Man beachte, daß 'enable secret [level NUM] ![level NUM] STRING' nicht funktionieren würde: Die Negation eines optionalen Patterns — hier '![level NUM]' — kann nie einen Match erzeugen, weil der optionale Teil '[…]' zumindest mit einem Text der Länge 0 immer einen Match liefert.

Copyright © Fraunhofer IESE 2003

Makros

Einige Konfigurationsklauseln des IOS sind äußerst komplex und variantenreich. Entsprechend länglich fällt das dazu passende Pattern aus. Um Textmuster möglichst übersichtlich spezifizieren zu können, ist die Verwendung von Makros vorgesehen. Ein Makro ist ein Paar (<Makroname>, <Makro-Pattern>), wobei der Makroname aus Großbuchstaben, Ziffern und '_'-Zeichen zu bilden ist, mit führendem Zeichen ungleich Ziffer. Nach Definition des Makros kann der Makroname fortan wie ein Atom verwendet werden. Wann immer in einer Pattern-Spezifikation '… <Makroname> …' verwendet wird, erfolgt an dieser Stelle stillschweigend eine Pattern-Substitution durch '{<Makro-Pattern>}'. Man beachte, daß jedes nicht-atomare Makro-Pattern vor Einfügung mit geschweiften Klammern umgeben wird. Dies stellt sicher, daß die Ersetzung wirklich als atomare Einheit angesehen wird und innerhalb von Komposita keine Verwirrung stiften kann. Ein Beispiel für ein Makro ist etwa

```
SUBNET => "{ IPADDR IPMASK | any | host IPADDR }"
```

Das Makro könnte in einem umfassenderen Pattern eingesetzt werden, um dessen Formulierung zu vereinfachen. So entspräche zum Beispiel

```
deny ip SUBNET SUBNET
```

dem ausgeschriebenen Pattern

```
deny ip {IPADDR IPMASK|any|host IPADDR}{IPADDR IPMASK|any|host IPADDR}.
```

Werden in einem Pattern Makros verwendet, so richtet sich die Numerierung bei der Pattern-Zerlegung (vgl. Abschnitt «Pattern-Zerlegung») nach der vollständig ausgeschriebenen Fassung des Textmusters. Dies stellt sicher, daß jeder Pattern-bestandteil — ob innerhalb oder außerhalb eines Makros — für Prüfmodule gezielt zugreifbar ist. So erhält im Beispiel oben das erste Schlüsselwort 'host' etwa die Nummer 2.3, das zweite Auftreten von 'host' die Nummer 3.3. Die Werte des kompletten 'SUBNET'-Makros lassen sich unter den Nummern 2 bzw. 3 zugreifen, ganz so, als sei 'SUBNET' ein vordefiniertes ATOM.

Makros dürfen auch verschachtelt werden, d.h. zur Definition eines Makros dürfen andere, zuvor bereits definierte Makros herangezogen werden. Es ist allerdings darauf zu achten, daß in einer Makrodefinition nicht auf das Makro selbst oder auf ein erst später definiertes Makro Bezug genommen wird.

Jedes Prüfmodul darf seine eigenen Makros definieren. Es spielt keine Rolle, ob mehrere Module den gleichen Makronamen mit unterschiedlichen Bedeutungen belegen. Alle Makros werden lokal, relativ zum definierenden Prüfmodul aufgelöst. So sind im Parser auf globaler Ebene die lokalen Makros nicht mehr

Copyright © Fraunhofer IESE 2003

von Bedeutung. Dies stellt sicher, daß durch das Hinzufügen oder Erweitern von Prüfmodulen keine Namenskollisionen verursacht werden.

Neben den lokalen Makro-Definitionen einzelner Prüfmodule besteht die Möglichkeit, global gültige Makros im Parser zu definieren. Dies ermöglicht es, eigene Atome von allgemeiner Nützlichkeit einzuführen, ohne dafür in die Abläufe des Parsers selbst eingreifen zu müssen. Global vereinbarte Makronamen dürfen lokal in den Prüfmodulen auf Wunsch umdefiniert werden. Lokal definierte Makros überschreiben dabei nicht nur globale Vordefinitionen, sondern gegebenenfalls sogar die Bezeichner der Standard-Atome (z.B. 'IPADDR', 'NUM', …), wovon allerdings dringend abzuraten ist. Derzeit sind folgende Makros global vordefiniert (aus Benutzersicht sind diese Makros kaum von Atomen zu unterscheiden):

```
IPMASK ::= IPADDR
IF_TYPE::= /(?i)[a-z]+/
IF_NUM ::= /\d+(:\d+|\/\d+(\.\d+)?)*/
```

Schnittstelle zum Hauptprogramm

Das Parser-Modul hat eine sehr schlanke Schnittstelle zum Hauptprogramm. Die einzigen nach außen sichtbaren Funktionen des Parsers sind 'new()', 'parse()' und 'analyze()'. Deren Verwendung geht aus dem Beispiel in Abbildung 3, Seite 19, hervor. Der Anwender muß sich über die Parser-Schnittstelle zum Hauptprogramm keine Gedanken machen, selbst wenn er neue Prüfmodule entwickelt und einfügt.

Schnittstelle zu den Prüfmodulen

Mit der Schnittstelle zwischen dem Parser und den Prüfmodulen kommt der Anwender normalerweise nicht in Berührung, selbst wenn er eigene Prüfmodule konstruiert. Das korrekte Zusammenspiel ergibt sich automatisch aufgrund der Vererbungsmechanismen.

Die Interaktion zwischen Parser und Prüfmodulen erfolgt nur in einer Richtung: Alle Aktivität geht vom Parser aus. Aus Sicht des Parsers sind alle Prüfmodule vom Objekttyp 'Module'. Daß es sich in Wahrheit um abgeleitete, verfeinerte Objekte handelt, ist an der Schnittstelle zwischen Parser und Prüfmodulen nicht erkennbar. Folglich nutzt der Parser auch nur die öffentlichen Funktionen des Objekttyps 'Module'. Prüfmodule erben diese Schnittstelle aufgrund ihrer Ableitung von 'Module' ohne weiteres Zutun. Im einzelnen handelt es sich um folgende Funktionen:

Copyright © Fraunhofer IESE 2003

- name()
  Der Aufruf $module->name() liefert den Typnamen des Prüfmoduls, hauptsächlich, um bei Fehlermeldungen präzise das Modul anzugeben, in dem das Problem aufgetreten ist.

- requires()
  Der Aufruf $module->requires() liefert eine Liste aller Prüfmodule zurück, von denen das mit $module bezeichnete Prüfmodul abhängig ist. Die Liste ist normalerweise leer. Bei der Entwicklung neuer Prüfmodule bietet sich aber die Gelegenheit, logische Abhängigkeiten zwischen einzelnen Prüfmodulen auf diesem Wege anzuzeigen. Bei Aktivierung eines Moduls werden alle als abhängig gekennzeichneten Module automatisch mitaktiviert.

- patterns()
  Der Aufruf $module->patterns() liefert alle Patterns, die der Parser im Auftrag des Moduls aufspüren und zurückmelden soll, sowie den Namen des jeweiligen Pattern Handlers, der bei Auftreten des Patterns die Weiterverarbeitung übernimmt.

- macros()
  Der Aufruf $module->macros() liefert alle Makros, die das Modul zur Formulierung seiner Patterns verwendet. Der Parser nutzt diese Information, um die Patterns in ihre ausführliche Form zu expandieren.

- postprocessing()
  Der Aufruf $module->postprocessing() liefert die Methode, mit der das Modul eine abschließende Gesamtwertung des Prüfbefunds ermittelt.

3.3 Prüfmodule

Ein Themenkomplex eines Sicherheitsaudits umfaßt in der Regel eine Reihe von Prüfpunkten. Logisch zusammengehörige Prüfpunkte werden in der Regel als eigenständiges Prüfmodul gebündelt.

Konstruktionsprinzip

Alle Prüfmodule sind von einem gemeinsamen Basistyp — der Objektklasse 'Module' — abgeleitet. Durch Vererbung sind sie mit allen wesentlichen Mechanismen ausgestattet, die erforderlich sind, um in das Werkzeug-Rahmenwerk zu passen (siehe Abschnitt «Schnittstelle zu den Prüfmodulen» oben). Aufgrund der Vererbung lassen sich neue Prüfmodule mit geringem Aufwand konstruieren und in das Werkzeug integrieren. Abbildung 4 zeigt den grundlegenden Aufbau eines Prüfmoduls. Sämtliche Prüfmodule basieren auf diesem Programmskelett.

Copyright © Fraunhofer IESE 2003

```
package Plugin::SomeModule;

# Vererbung: Jedes Prüfmodul ist ein («is a») Plugin::Modul
# Damit ist die Schnittstelle zum Parser automatisch verfügbar
use Plugin::Module;
use vars qw(@ISA);
@ISA = ("Plugin::Module");

# Private Makro-Definitionen des Prüfmoduls
my $macroref = [
  SUBNET => "{ IPADDR IPMASK | any | host IPADDR }",
  …
];

# Pattern-Definitionen, an denen das Prüfmodul interessiert ist
# mit Zuordnung der Liste aller lokal zuständigen Pattern Handler
my $patternref = {
  "line [aux|con|console|tty|vty] NUM [NUM]" =>
     { HANDLER => [ \&_line_handler ], },
  …
};

# Konstruktor für Instanzen des Prüfmoduls
sub new {
  my $proto = shift;
  my $class = ref($proto) || $proto;
  my $self;

  $self=$class->SUPER::does_exist();
  if (!defined $self) {
    $self = $class->SUPER::new();
    $self->{MACROS} = $macroref;
    $self->{PATTERNS} = $patternref;
    $self->{POSTPROCEDURE} = \&_final_conclusion;
    }
  return $self;
}

# Handler-Methoden: Für jedes Pattern wenigstens ein Handler
sub _line_handler {
  my ($self, $match) = @_;
  …
}
…

# Nachverarbeitungs-Methode zur Aufbereitung der Gesamtbewertung
sub _final_conclusion {
  my ($self) = @_;
  …
}
```

Abbildung 4 Konstruktionsprinzip der Prüfmodule

Copyright © Fraunhofer IESE 2003

Um das Werkzeug um spezifische Prüfschritte zu erweitern, erstellt der Anwender ein neues Prüfmodul gemäß Abbildung 4. Der Zuschnitt auf bestimmte Prüfaufgaben erfordert folgende Maßnahmen:

- In der Kopfzeile vergibt der Anwender einen Namen für das neue Prüfmodul.
- Im Abschnitt «Makro-Definitionen» kann er lokale Makros deklarieren, um die Spezifikation der zu prüfenden Patterns zu vereinfachen.
- Im Abschnitt «Pattern-Definitionen» legt der Anwender fest, welche Textmuster analysiert werden sollen und welche Handler für jedes Pattern zuständig sind. Ein Pattern kann auch mehrere Handler zugleich bedienen.
- Im Abschnitt «Handler-Methoden» ist für jeden deklarierten Pattern Handler die entsprechende Prüfroutine zu schreiben. Der angegebene Code wertet alle Textfragmente, die dem Pattern entsprechen, aus und bildet daraus eine Sicherheitsbewertung (mehr dazu im Unterkapitel «Pattern Handler» unten).
- Im Abschnitt «Nachverarbeitungs-Methode» wird, soweit erforderlich, eine Routine bereitgestellt, die aus der Summe aller dem Modul gemeldeten Textfragmente und den daraus gewonnenen Prüfinformationen eine abschließende Gesamtbewertung bildet. Diese Methode darf auch undefiniert bleiben.

Die genannten Anpassungen reichen aus, um ein funktionstüchtiges Prüfmodul zu erstellen. Man beachte, daß der Anwender die Abschnitte «Vererbung» und «Konstruktor» in der Regel nicht antasten muß. Diese Passagen können zunächst unverändert von der Codeschablone laut Abbildung 4 übernommen werden. Im Kern schrumpft somit die Anleitung für die Konstruktion neuer Prüfmodule auf zwei Punkte zusammen:

1. Spezifiziere die Patterns, die analysiert werden sollen.
2. Schreibe pro Pattern einen Handler, der die vorgesehenen Analysen an den gefundenen Textfragmenten vornimmt sowie bei Bedarf einen Nachbearbeitungs-Handler, der alle gesammelten Daten am Ende einer abschließenden Gesamtanalyse unterzieht.

Pattern Handler

Pattern Handler (kurz: Handler) sind Methoden des Prüfmoduls, die vom Parser aufgerufen werden, sooft ein zugeordnetes Text-Pattern in der Router-Konfigurationsbeschreibung erkannt wird. Der Parser versorgt den Handler bei jedem Aufruf mit zwei Parametern:

Copyright © Fraunhofer IESE 2003

- Der Parameter '$self' verweist auf das Prüfmodul und ermöglicht es der Methode, bei Prüfschritten auf beliebige private Prüfmoduldaten zurückzugreifen. Die Methode ist so in der Lage, Informationen über mehrere Aufrufe hinweg zu speichern oder mit anderen Handlern des Prüfmoduls Erkenntnisse auszutauschen.

- Der Parameter '$match' verweist auf eine Attribute-Werte-Tabelle, die alle wesentlichen Informationen zum Pattern und dem dazu passenden Textfragment enthält, darunter:

  – LINE_NO - die Zeilennummer, in der das Pattern aufgetreten ist

  – LINE - der Wortlaut der Zeile

  – MATCH - der Teil des Wortlauts, der dem Pattern entspricht

  – PATTERN - das Pattern, welches die Erkennung ausgelöst hat

  – EXPANSION - die Langform des Patterns nach Auflösung aller Makros

  – NEGATION - ein Flag für das Auftreten von '**no** <PATTERN>'

  – *<Fragmentnr.>*- die Bestandteile des Textfragments, zerlegt nach Fragmentnummern (vgl. Abschnitt «Pattern-Zerlegung», Seite 22).

Mit Hilfe der zwei Parameter hat der Pattern Handler Zugriff auf alle relevanten Informationen, die für eine Prüfung benötigt werden. Abbildung 5 zeigt den Ausschnitt eines typischen Datensatzes, wie ihn der Parser mittels '$match' an einen Pattern Handler übergibt.

```
0:              'interface'
1:              'Loopback0'
1.0:            'Loopback'
1.1:            '0'
CRITICALITY:    'INFO'
EXPANSION:      'interface
                 {/(?i)[a-z]+/ /\d+(:\d+|\/\d+(\.\d+)?)*/}'
LINE:           'interface Loopback0'
LINE_NO:        '22'
MATCH:          'interface Loopback0'
PATTERN:        'interface {IF_TYPE IF_NUM}'
```

Abbildung 5 Typisches Beispiel für einen '$match'-Datensatz, wie ihn der Parser an einen Handler übergibt

Copyright © Fraunhofer IESE 2003

Über den Verweis '$match' läßt sich leicht feststellen, welches Pattern den Aufruf ausgelöst hat (ein Handler kann potentiell mehrere Pattern bedienen). Der genaue Inhalt der erkannten Router-Konfigurationsanweisung und seine Position in der Konfigurationsbeschreibung sind zugreifbar.

Der Verweis '$self' ermöglicht den Rückgriff auf Kontextwissen bei der Beurteilung des Textfragments. Darüber hinaus bietet er Zugang zu allen (von Module ererbten) Unterstützungsfunktionen, die das Prüfmodul zur Durchführung von Prüfschritten bereithält.

Mit den verfügbaren Mechanismen können Prüfschritte in einfacher Weise als PERL-Programmcode implementiert werden, ohne daß der Anwender genauere Kenntnisse über das Zusammenspiel von Hauptprogramm, Parser, Prüfmodulen und Ein-/Ausgabe benötigt. Kapitel 8 illustriert an einem Beispiel, wie bei der Erstellung eines neuen Prüfmoduls vorzugehen ist.

Modulparametrisierung

Je nach Prüfaufgabe eines Prüfmoduls kann es sinnvoll sein, bestimmte Prüfschritte zu parametrisieren. Das Prüf-Rahmenwerk sieht daher vor, einem Prüfmodul bei Bedarf eine oder auch mehrere Konfigurationsdateien zuzuordnen.

Inhalt und Format solcher Konfigurationsdateien sind nicht vorgegeben. Jedem Prüfmodul können individuell strukturierte Konfigurationsinformationen zugeordnet werden. Aufgrund fehlender Formatfestlegungen gibt es derzeit keine standardisierten Zugriffsmethoden auf Konfigurationsdateien. Es obliegt dem Programmierer des jeweiligen Moduls, Code für das Öffnen und Auslesen der Konfigurationsdatei bereitzustellen. Der Quellcode der Standardmodule 'Passwords', 'IngressEgress' und 'CompoundPatterns' liefern mögliche Vorlagen für den sinnvollen Einsatz von Konfigurationsdateien auf unterschiedlichem Anspruchsniveau (siehe Kapitel 4).

Um die Übersichtlichkeit des Prüfwerkzeugs zu gewährleisten, sollten alle Konfigurationsdateien im eigens dafür vorgesehenen Konfigurationsverzeichnis (siehe Verzeichnisstruktur in Kapitel 3.7) unter dem Namen <*Prüfmodul-Name*>.conf (bzw. <*Prüfmodul-Name*>.conf<*Nr.*> bei der Verwendung mehrerer Konfigurationsdateien) angelegt werden.

3.4 Befunddatenbank

Alle Prüfmodule speichern ihre Befunde in strukturierter Form in einer Ergebnisdatenbank des Typs 'ResultData' ab. Die Datenbank bildet die Grundlage, um am Ende den Gesamtbefund des Prüflaufs aufzubereiten. Der Einsatz einer

Copyright © Fraunhofer IESE 2003

Datenbank entkoppelt die Prüfmodule von spezifischen Darstellungsformaten der Ausgabe.

Logisches Modell der Befunddaten

Prüfmodule können Befunddaten unterschiedlicher Art in der Datenbank ablegen. Möglich sind

- Anmerkungen
- Befundkategorien
- Verbesserungsvorschläge
- Dumps
- Verweise
- Lesezeichen

Anmerkungen sind Zeichenketten mit einem Kommentar zum Sicherheitsstatus einer Konfigurationseigenschaft. Jede Anmerkung ist mit einer Kritikalitätseinstufung versehen, die von 'OKAY', 'INFO', 'CHECK' über 'WARN' bis 'ALERT' reichen kann. Die Einstufung gibt an, ob es sich bei der Anmerkung um einen positiven, neutralen, unklaren, negativen oder gar kritischen Befund handelt. Die Kritikalität eines Befundes wird in der Ausgabe durch entsprechende Färbung kenntlich gemacht.

Befundkategorien bieten die Möglichkeit, Befunde schlagwortartig bestimmten Themenbereichen zuzuordnen. Der Benutzer kann den Namen und die Bedeutung einer solchen Kategorie selbst festlegen und jeden Befund einer Befundkategorie zuordnen. Solche Kategorien ermöglichen es später in der Ausgabe, einzelne Analysesichten auf ausgewählte Problembereiche einzuschränken. Beispiele für mögliche Kategorien sind etwa 'Benutzerauthentisierung', 'Logging' oder auch 'Accounting'.

Verbesserungsvorschläge bieten die Möglichkeit, zu einem Befund einen «Bug Fix» mit anzugeben. In der Regel wird es sich dabei um IOS-Konfigurationsbefehle handeln, deren Funktion zusätzlich mit einem Kommentar erläutert werden kann. Es steht dem Anwender jedoch frei, als Verbesserungsvorschlag einen gewöhnlichen Klartext zu hinterlegen.

Dumps dienen dazu, umfassendere Analyseergebnisse tabellarisch — zeilenweise und mit unterschiedlichen Einrücktiefen — im Textformat auszugeben.

Copyright © Fraunhofer IESE 2003

Dumps werden benötigt, wenn ein Befund den Rahmen einer kurzen Anmerkung sprengt. In der Ausgabe sind Hyperlinks auf Dumps möglich.

Verweise dienen dazu, Querbeziehungen zwischen einzelnen Befunden und bei Bedarf auch zu externen Informationsquellen — etwa zum IOS Masterindex von Cisco — herzustellen. In der Ausgabe werden solche Verweise in der Regel als Hyperlinks dargestellt, denen der Anwender per Mausklick nachgehen kann. Referenzierbar sind zum Beispiel Konfigurationszeilen, Dumps oder externe Internet-Seiten.

Lesezeichen (Bookmarks) repräsentieren keine eigenständigen Befunde. Sie sind lediglich ein Strukturierungsmittel, um Verweise zu logischen Gruppen zusammenzufassen. Öffnet man eine Bookmark-Kategorie in der Ausgabe, so erscheint eine Auswahl von Verweisen zu dieser Kategorie. Der Anwender kann beliebige Bookmark-Kategorien einführen und einzelne Verweise beliebigen — auch mehreren — Kategorien zuordnen. Siehe dazu auch Abbildung 6 auf Seite 36.

All diese Befundarten können mit unterschiedlichem Geltungsbereich in der Datenbank abgelegt werden:

- Befunde zu einzelnen Konfigurationsanweisungen
- Befunde zu Klauselkontexten
- Befunde zu logischen Kontexten

Befunde zu Konfigurationsanweisungen beziehen sich auf eine einzelne Klausel der Router-Konfigurationsbeschreibung. Ein Prüfmodul kann einer Zeilennummer Anmerkungen, Verweise oder Verbesserungsvorschläge in beliebiger Zahl zuordnen. Durch Anklicken der Konfigurationszeile kann der Anwender alle zugeordneten Befunde anzeigen lassen.

Befunde zu Klauselkontexten beziehen sich auf die sogenannten "Configuration Modes" des Router-Betriebssystems IOS. Configuration Modes bilden einen Kontext, in dem ganz bestimmte Routereinstellungen vorgenommen werden können. Beispiele sind etwa der "Interface Mode" oder der "Line Mode". Viele Konfigurationsanweisungen sind nur in ganz bestimmten Kontexten erlaubt, und ein Konfigurationsfehler in einem Kontext hat Auswirkungen auf alle Einstellungen im gleichen Kontext. Prüfmodule haben die Möglichkeit, alle Zeilen eines solchen Kontexts zu einen Klauselkontext[7] zusammenzufassen und ihnen

7 Noch passender wäre stattdessen die Bezeichnung «Zeilenkontext», doch leider würde dies zu in der englischen Übersetzung zu Verwechslungsgefahr mit der von IOS bereits vergebenen Bezeichnung «Line» für eine serielle Schnittstelle führen, weshalb wir uns für «Klausel» entschieden haben.

Copyright © Fraunhofer IESE 2003

gemeinsame Anmerkungen, Verbesserungsvorschläge und Verweise zuzuordnen. Der Klauselkontext mit allen zugehörigen Zeilen und Befunden wird in der Ergebnisdarstellung als zusammenhängende Einheit präsentiert. Die Kritikalität eines Klauselkontexts ergibt sich aus der Kritikalität aller seiner Anmerkungen sowie der Kritikalität der zugehörigen Konfigurationszeilen. Der Configuration Mode eines IOS-Befehls legt eindeutig dessen Klauselkontext fest.

Darüber hinaus besteht jedoch die Möglichkeit, Zeilen zu beliebigen logischen Kontexten zusammenzufassen und diesen Kontexten ebenfalls Anmerkungen, Verbesserungsvorschläge und Referenzen zuzuordnen. So kann man beispielsweise die Deklarationsklauseln einer Access-Liste und alle Klauseln, in denen diese Access-Liste referenziert wird, zu einem globalen «access-list»-Kontext zusammenfassen. Alle Konfigurationsfehler mit Bezug zu der Access-Liste können so zusammenhängend erfaßt und dargestellt werden.

Eine Zeile der Konfigurationsdatei kann beliebig vielen logischen Kontexten zugeordnet werden, aber — durch IOS vorgegeben — höchstens einem Klauselkontext.

Schnittstelle zu den Prüfmodulen

Prüfmodule nutzen die Datenbankschnittstelle in der Regel nicht direkt, sondern mittels Wrapper-Methoden, die von der gemeinsamen Basisklasse aller Prüfmodule — Plugin::Module — mit verkürzter Parameterliste bereitgestellt werden. Die verfügbaren Methoden beschreibt Anhang A.

Speichern und Laden der Befunddatenbank im XML-Format

Aus den Befunddaten können — unabhängig vom Prüflauf — verschiedene Ergebnisdarstellungen generiert werden. Zu diesem Zweck wird die Befunddatenbank im XML-Format als Datei abgespeichert. Die XML-Datei kann später jederzeit wieder von eingelesen und in unterschiedlichen Darstellungsformaten angezeigt werden. Dazu verfügt der Werkzeugprototyp über das Darstellungswerkzeug 'make_html'. Das Programm findet sich im Utils-Unterverzeichnis und generiert eine komplexe HTML-Ansicht, bestehend aus mehreren Ergebnis-Frames. Weitere Werkzeuge für andere Ausgabeformate — zum Beispiel MS Excel-Format — sind zukünftig denkbar, derzeit jedoch noch nicht verfügbar.

3.5 Ausgabe des Analysebefunds

Die Ergebnisdatenbank speichert alle Befunde in einem internen Format. Es gibt verschiedene Möglichkeiten, daraus eine akzeptable Darstellung zu erzeugen.

Copyright © Fraunhofer IESE 2003

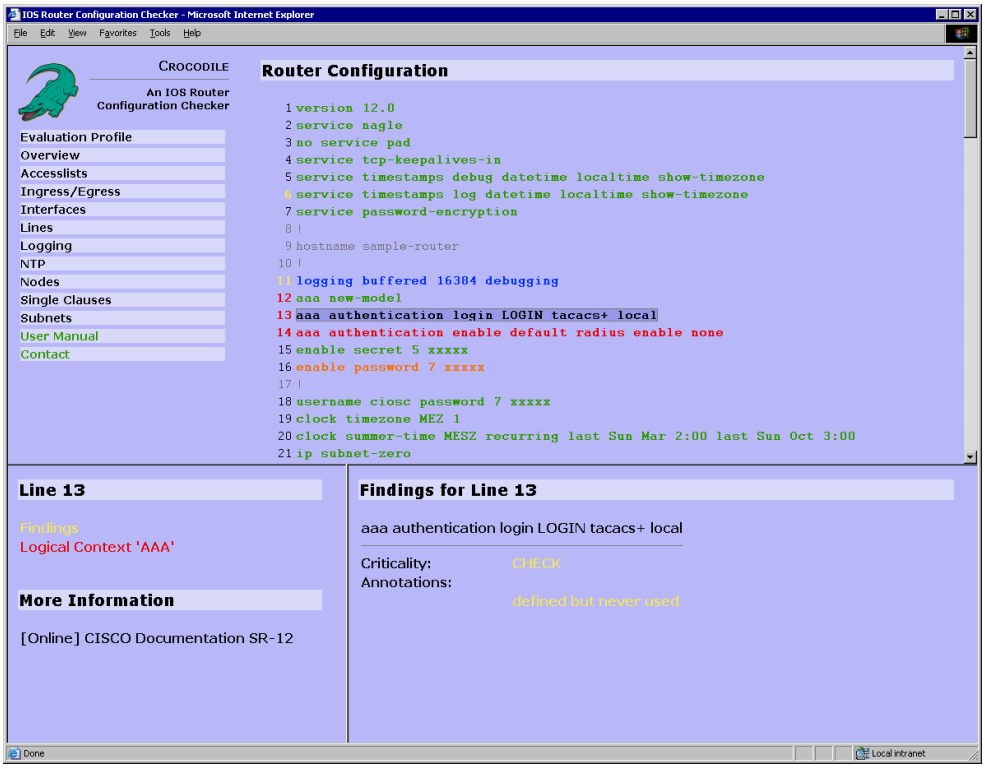

Abbildung 6 Detaillierte Ergebnisdarstellung im Hypertext-Format

Hypertext-Ausgabe

Das Prüfwerkzeug unterstützt derzeit eine HTML-basierte Darstellungsvariante, die unterschiedliche Sichten in mehreren, untereinander verknüpften Frames darstellt. Mittels Hyperlinks kann der Anwender in den Befunddaten auf unterschiedlichen Abstraktionsniveaus navigieren. Abbildung 6 zeigt ein typisches Beispiel. Im oberen Fenster der Darstellung wird die untersuchte Konfigurationsdatei angezeigt. Einzelne Konfigurationsklauseln sind je nach Schwere des Befunds eingefärbt. Jede Klausel ist außerdem mit einem Hyperlink hinterlegt; durch Anklicken werden in den beiden unteren Fenstern Detailinformationen zur jeweiligen Klausel (im Beispiel die Klausel in Zeile 13 des Konfigurationstexts) angezeigt. In der linken unteren Hälfte werden, nachdem auf eine Zeile geklickt wurde, zugehörige Verweise als Hyperlinks angezeigt. Durch Klick auf einen dieser Links wird im unteren Teil rechts die gewünschte Information angezeigt. Auf Wunsch kann man alle Klauseln eines Kontexts und den Kontext-Prüfbefund durch Anklicken in die Anzeige holen. Man kann auch unmittelbar zu ausgewählten Seiten des IOS Reference Manuals im Internet wechseln — passende Verweise werden vom Werkzeug automatisch in der Anzeige bereitgestellt.

Copyright © Fraunhofer IESE 2003

Zusätzlich umfaßt die Anzeige noch einen 'Bookmark'-Bereich. Dort sind Verweise auf vertiefende Analysen, Hintergrundinformationen sowie auf das Benutzerhandbuch hinterlegt. Einzelne Analysemodule können nach Bedarf weitere Bookmarks anlegen, um logisch zusammenhängende Befunde übersichtlich zu gliedern. Das Werkzeug bietet dafür eine einfache Programmierschnittstelle für Plugins (siehe Anhang A).

HTML-Übersichten

Die HTML-Darstellung ist facettenreich, erfordert aber auch einiges Navigieren, um die Summe aller Befunde zu erschließen. Daher findet sich im 'Bookmark'-Bereich der Verweis «Evaluation Report» auf eine Befundzusammenfassung. Es handelt sich um ein HTML-Dokument, das die wesentlichen Anmerkungen zur analysierten Routerkonfiguration auf einer einzigen Seite kompakt und im Zusammenhang darstellt. Der Anwender kann sich auf diesem Wege zuerst einen Überblick verschaffen. Abbildung 7 zeigt typische Ausschnitte aus einer solchen Übersichtsdarstellung. Es handelt sich um den Konfigurationstext, bei dem die einzelnen Klauseln je nach Prüfbefund eingefärbt sind. Anmerkungen zu den Klauseln sind als Kommentare, mit '***' gekennzeichnet, in den Konfigurationstext eingefügt.

Teil dieser Übersichtsdarstellung ist das «Evaluation Profile» (Ausschnitt unten rechts in Abbildung 7). Diese Ansicht kann der Anwender über den gleichnamigen Bookmark-Eintrag auch separat über das CROCODILE Hauptfenster anwählen. Die Ansicht liefert auf einen Blick die wichtigsten Informationen zum Untersuchungsgegenstand und dem Prüfergebnis. Anzahl und Schwere der Befunde werden — geordnet nach Prüfmodulen — als Business-Graphik in Form von Balkendiagrammen dargestellt. Die Länge der Balken korreliert mit der Menge an kritischen Befunden in den Kategorien 'OKAY', 'INFO', 'CHECK', 'WARN' und 'ALERT', die das jeweilige Prüfmodul ermittelt hat. Der Anwender sieht auf einen Blick, welches die größten Schwächen der Konfigurationsbeschreibung sind. Durch Mausklick in die Balken des Diagramms kann sich der Anwender direkt die entsprechenden Befunde anzeigen lassen. Abbildung 8 zeigt ein Evaluation Profile sowie die Anzeige einer bestimmte Befundkategorie, die man durch Anklicken des Diagramms in die Anzeige holen kann.

Differentielle Ergebnisdarstellung

Wird eine Routerkonfiguration geändert, so sollte sie danach einer erneuten Prüfung unterzogen werden. Dabei interessiert den Anwender weniger die Summe aller Prüfbefunde, sondern vor allem, wie sich die Konfigurationsänderungen in Änderungen des Prüfergebnisses niederschlagen. Das Werkzeug bietet die Möglichkeit einer vergleichenden Ergebnisdarstellung, relativ zu einem früheren Prüfbefund: In dieser Ansicht werden nur solche Befunde farbig hervor-

Copyright © Fraunhofer IESE 2003

gehoben, die sich seit der letzten Prüfung geändert haben. Alle übrigen Befunde, die in beiden Evaluationsläufen gleich sind, werden in neutralem schwarz dargestellt. In diesem Aufbereitungsmodus stechen Änderungen im Prüfergebnis sofort ins Auge.

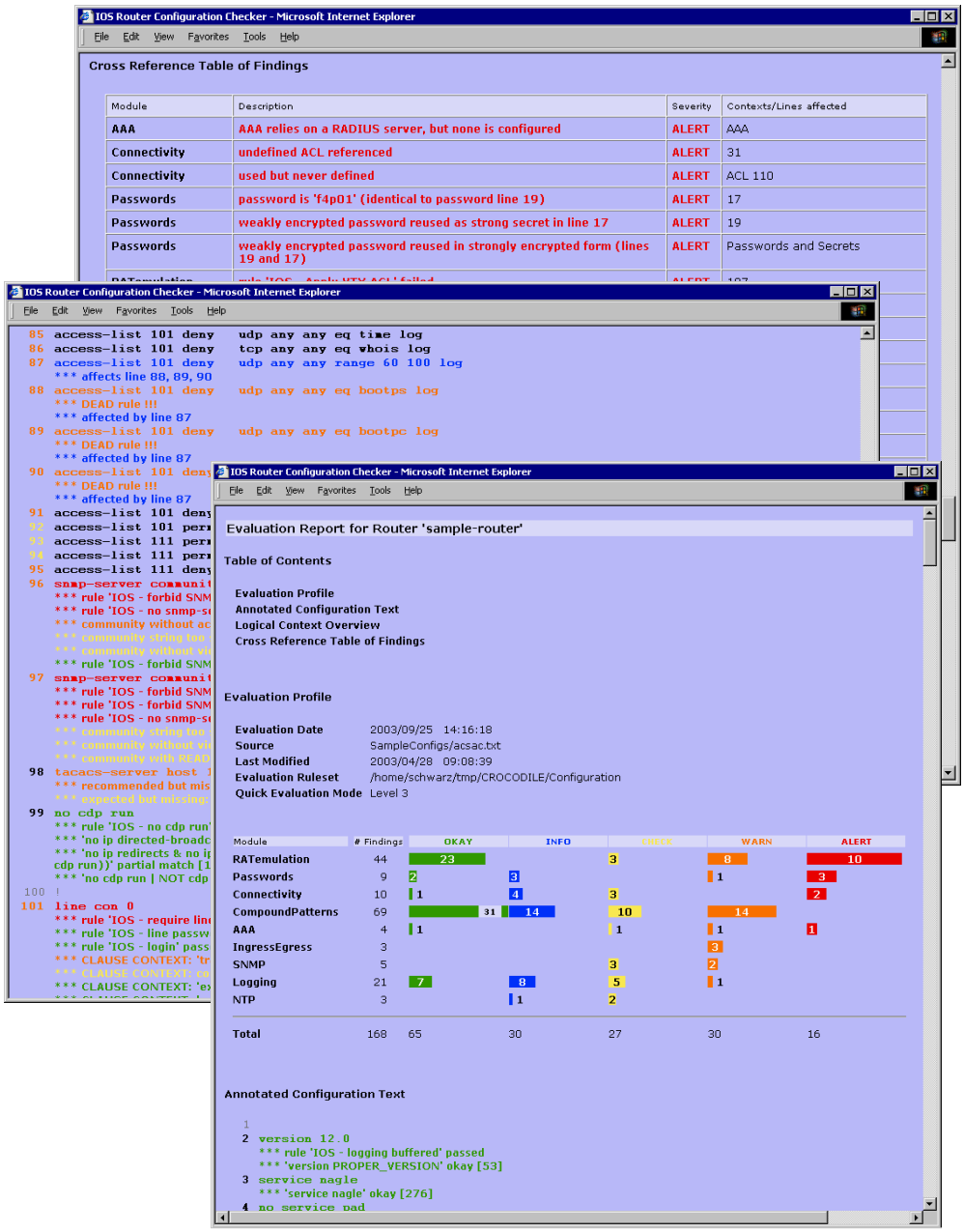

Abbildung 7 Typische Ergebnisübersicht

Copyright © Fraunhofer IESE 2003

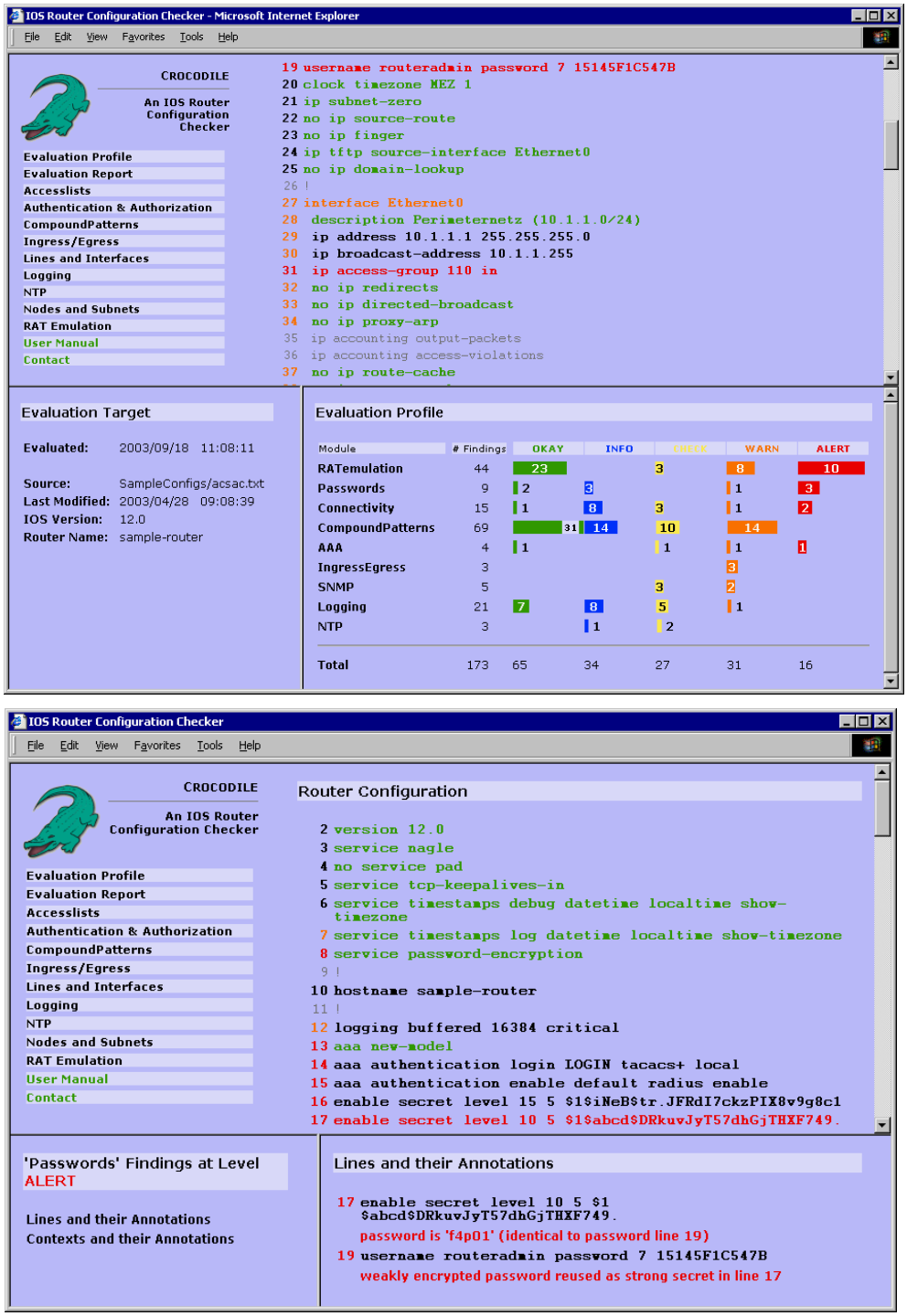

Abbildung 8 «Evaluation Profile»-Ansicht (oben) und Detailansicht der Befundkategorie 'ALERT' des Prüfmoduls 'Passwords' (unten), die man durch Anklicken der entsprechenden Farbfläche im Balkendiagramm erhält.

Copyright © Fraunhofer IESE 2003

Näheres dazu findet sich in Kapitel 7.2. Eine Beispiel für eine differentielle Ergebnisaufbereitung zeigt Abbildung 28 auf Seite 113.

Konfigurieren der Darstellungsparameter

Das Ausgabemodul für die HTML-Darstellung ist konfigurierbar. In der Datei 'Output/HTML_output.conf' können Hintergrund- und Schriftfarbe festgelegt werden, außerdem die Zuordnung von Farben zu Kritikalitätsstufen. Abbildung 9 zeigt die Konfigurationsdatei mit typischen Eintragungen. Die Bedeutung

```
## HTML_output.conf
## Config-file for HTML output

## Some general settings about the appereance
show_not_recognized    = YES
resize_allowed         = YES
frame_borders          = YES
main_window_percentage = 60%
link_window_percentage = 35%

## Settings about the style
# You may define a font size, but you may also avoid it
# and just rely on the browsers default settings
# font_size          = 12pt
font_family          = Verdana,Arial,arial,Helvetica,helvetica
font_family_listing  = Courier,courier,monospace
font_size_heading    = 120%
font_size_listing    = 110%
font_size_bookmark   = 90%
font_weight          = bold
font_color           = #000
bg_color             = #BBF
heading_bgcolor      = #DDF
margin               = 1px

## Define the RGB-Mix for each severity
colorcode {
        OKAY      = #390
        INFO      = #03F
        CHECK     = #FE5
        WARN      = #F70
        ALERT     = #E00
        UNDEF     = #777
        SOLID     = #000
}
```

Abbildung 9 Typische Einstellungen für die HTML-Ausgabe in 'Output/HTML_output.conf'

Copyright © Fraunhofer IESE 2003

einzelner Attribute ist meist unmittelbar aus dem Attributnamen ableitbar, mit folgenden Ausnahmen.

Das Attribut 'show_not_recognized' gibt an, ob der untersuchte IOS-Konfigurationstext vollständig ausgegeben wird, oder ob Konfigurationsklauseln, die das Werkzeug in der Analyse nicht berücksichtigt hat, ausgeblendet werden sollen. Das Attribut 'resize_allowed' bestimmt, ob die Aufteilung der Browser-Fensters in einzelne Frames unverrückbar sein soll, oder ob der Anwender die Größe der Frames mittels Mausbewegung verändern darf. Mit Hilfe des Attributs 'frame_borders' kann der Anwender die Darstellung der Frame-Umrandung auf Wunsch unterdrücken. Das Attribut 'main_window_percentage' gibt an, welcher Anteil der insgesamt verfügbaren Fensterhöhe des Browsers für den Haupt-Frame zur Darstellung des Konfigurationstexts verwendet werden soll; der Rest wird für die Frames 'Links' und 'Display' verwendet, um Detailinformationen darzustellen. Das Attribut 'link_window_percentage' bestimmt, welcher Anteil der insgesamt verfügbaren Fensterbreite des Browsers für den 'Links'-Frame reserviert wird; der Rest wird für den 'Display'-Frame zur Darstellung von Detailinformationen verwendet.

Abgesehen von der Einstellung einer individuell bevorzugten Schriftgröße besteht normalerweise wenig Bedarf, die vorgegebenen Darstellungsparameter zu ändern. Anpassungen können jedoch erforderlich werden, wenn veraltetes Ausgabegerät eingesetzt werden soll oder wenn CROCODILE-Screenshots (z.B. in schwarz-weiß) weiterverarbeitet werden müssen.

3.6 Unterstützungsmodule

Bei komplizierten Prüfvorgaben kann die Programmierung eines Handlers erheblichen Aufwand erfordern. Das Werkzeug-Rahmenwerk sieht daher vor, häufig genutzte Funktionalität in separaten Unterstützungsmodulen zu kapseln, die allen Prüfmodulen als «Arbeitshilfen» verfügbar gemacht werden. Jedes neu hinzugefügte Prüfmodul kann bei der Implementierung seiner Pattern Handler auf einen Fundus von Hilfsfunktionen zurückgreifen, der eine kompakte, übersichtliche Formulierung des Prüfprogramms ermöglicht. Das Unterverzeichnis 'Auxiliary' dient zur Ablage der von den Prüfmodulen benötigten Unterstützungsmodule.

Persistente Daten

Einige Prüfschritte erfordern die Auswertung mehrerer Router-Konfigurationsanweisungen, also die Verknüpfung der Resultate mehrerer Pattern Handler, eventuell sogar mehrerer Prüfmodule. Solche Analysen sind nur möglich, wenn Informationen über die Aktivierungsdauer einzelner Pattern Handler hinaus gespeichert und bereitgehalten werden. Die entsprechenden Daten sind somit

Copyright © Fraunhofer IESE 2003

persistent in dem Sinne, daß sie unabhängig von der Lebensdauer einzelner Prüfmodule weiter existieren, bis das Werkzeug den endgültigen Prüfbefund ermittelt hat.

Das Werkzeug-Rahmenwerk sieht vor, persistente Daten durch Unterstützungsmodule zu repräsentieren. Jedes Unterstützungsmodul stellt die Abstraktion eines logischen Objekts dar und bietet Funktionen, um relevante Objekteigenschaften zu speichern oder abzufragen, Beziehungen zwischen Objekten zu modellieren oder um Operatoren auf das Objekt anzuwenden. Wir empfehlen die Verwendung solcher Unterstützungsmodule als allgemeines Mittel zur Strukturierung persistenter Daten.

3.7 Verzeichnisstruktur

Der Quellcode des Werkzeugs ist in verschiedene Unterverzeichnisse untergliedert. Die Strukturierung orientiert sich an den PERL-Namenskonventionen: In PERL werden Modul-Namenshierarchien 1:1 in entsprechende Verzeichnishierarchien abgebildet. Abbildung 10 zeigt die resultierende Verzeichnisstruktur.

Copyright © Fraunhofer IESE 2003

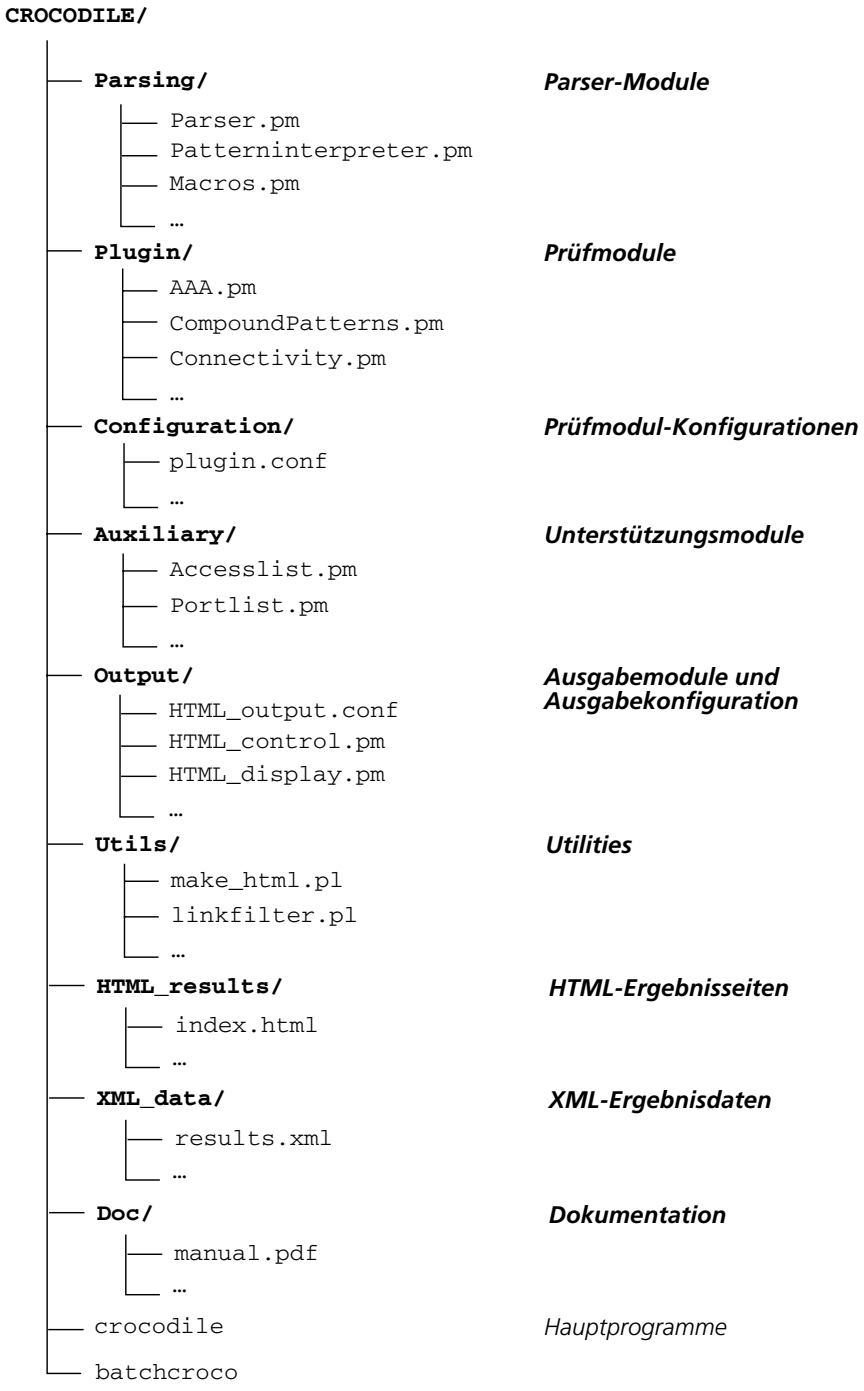

Abbildung 10 Verzeichnisstruktur der Werkzeug-Software

Copyright © Fraunhofer IESE 2003

Copyright © Fraunhofer IESE 2003

4 Prüfmodule

Zum Standardumfang des Prüfwerkzeugs gehören mehrere Prüfmodule. Sie realisieren bereits wesentliche Prüfschritte für IOS-Konfigurationen. Zusätzliche Module können leicht ergänzt werden, um den Prüfumfang des Werkzeugs flexibel zu erweitern. Im folgenden stellen wir die verfügbaren Prüfmodule vor.

4.1 Auswahl von Prüfmodulen zur Ausführung

Der Benutzer kann auswählen, welche der verfügbaren Prüfmodule bei einem Prüflauf tatsächlich zur Anwendung kommen sollen. Die Liste der zu aktivierenden Module steht in der Datei './Configuration/plugin.conf'. Fehlt diese Datei oder ist sie leer, so werden sämtliche unter './Plugin' verfügbaren Module geladen und aktiviert.

Durch Löschen (oder besser durch Auskommentieren) einzelner Einträge kann der Anwender den Prüfumfang des Werkzeugs reduzieren. Abbildung 11 zeigt eine Beispielkonfiguration, bei der die Module 'NTP.pm' und 'RATemulation.pm' abgeschaltet sind.

```
## plugin.conf -- Which plugin modules are selected for execution?
##
## If this file exists, only those plugins named here are loaded and
## executed. Plugins are specified one or multiple per line.
## Leading and trailing whitespace is ignored. A trailing '.pm'
## is optional. Specifications are case-insensitive. Module names
## not found in './Plugin' are ignored.

AAA
CompoundPatterns
Connectivity
IngressEgress
Logging
# NTP
Passwords
SNMP
# RATemulation
```

Abbildung 11 Konfigurationsbeispiel für 'plugin.conf' zur Festlegung der auszuführenden Prüfmodule

Copyright © Fraunhofer IESE 2003

4.2 Modul 'CompoundPatterns'

Das Prüfmodul 'CompoundPatterns' ist in der Lage, alle Prüfpunkte abzudecken, die sich durch den Nachweis einfacher Text-Patterns in der Konfigurationsbeschreibung des Routers entscheiden lassen. Sicherheitsmerkmale, die nur von bestimmten Kombinationen einzelner Konfigurationsanweisung ohne Bezug zur genauen Klauselausprägung abhängen, sind damit überprüfbar. Ein erheblicher Anteil aller Router-Konfigurationsmerkmale zählt zu dieser Kategorie.

Konfigurieren von Prüfvorgaben

'CompoundPatterns' ist ein universell einsetzbares Prüfmodul ohne fest vorgegebene Prüfvorgaben. Den konkreten Prüfauftrag erhält das Modul über die Modul-Konfigurationsdatei 'CompoundPatterns.conf' (vergleiche Verzeichnisstruktur in Abbildung 10, Seite 43).

Dieses Modul illustriert, wie man sehr flexibel nutzbare Prüfmodule konstruieren kann, indem man sie mittels einer Prüfmodul-Konfigurationsdatei parametrisierbar macht. Die genaue Syntax, in der Prüfaufträge spezifiziert werden, bestimmt das Prüfmodul, wobei es auf die verfügbaren Mechanismen des Werkzeug-Rahmenwerks, insbesondere Funktionen für Pattern Matching, zurückgreifen kann.

Die Konfigurationsdatei 'CompoundPatterns.conf' kann drei Arten von Spezifikationen enthalten:

- Makro-Definitionen
- Pattern-Definitionen (elementare oder zusammengesetzte Patterns)
- Kontext-Definitionen

Makros

Makro-Definitionen entsprechen in Funktion und Gestalt den üblichen Makros, wie sie auch innerhalb eines Prüfmoduls im Abschnitt «Makros» deklariert werden können (vgl. Abbildung 4, Seite 29). Sie ermöglichen es, komplexe Pattern-Bestandteile zur besseren Lesbarkeit der Patterns mit einem symbolischen Namen zu versehen. Der Makroname kann später als Atom zur Konstruktion umfassenderer Pattern-Spezifikationen genutzt werden. Das genaue Format einer Makrodefinition ist

```
<MAKRO NAME> ::= <basic pattern>
```


Copyright © Fraunhofer IESE 2003

wobei der Makroname aus dem Zeichenvorrat Großbuchstaben, Unterstrich und Bindestrich zu bilden ist. Zwei Beispiele sind etwa:

```
PROPER_VERSION  ::= '12.0' [ '.' NUM ]*
BAD_VERSION     ::= ! PROPER_VERSION NUM [ '.' NUM ]*
```

Die Definitionen besagen, daß jede Versionsnummer, die mit '12.0' beginnt und gegebenenfalls von weiteren, durch '.' abgetrennten Versionsziffern gefolgt wird, eine "PROPER_VERSION" im Sinne der Definition ist. Als "BAD_VERSION" wird jede Zeichenfolge angesehen, die sich aus Nummern — getrennt durch '.' — zusammensetzt, jedoch nicht mit einer PROPER_VERSION beginnt. Demnach wäre zum Beispiel '12.0.3.34' eine PROPER_VERSION, '12.3' hingegen eine BAD_VERSION. Ein Beispiel für die Verwendung der oben definierten Makros findet sich im folgenden Abschnitt.

Man beachte, daß in einer Makro-Definition nur ein elementares Pattern verwendet werden darf. Die Verwendung zusammengesetzter Patterns — d.h. die Verknüpfung elementarer Patterns mittels AND, OR, NOT, XOR usw. (siehe unten) — ist innerhalb von Makros nicht zulässig und wird von CROCODILE zurückgewiesen.

Patterns

Patterns bezeichnen Textmuster in der Router-Konfigurationsbeschreibung, deren Vorhandensein oder Fehlen zu überprüfen ist. Funktion und Gestalt eines Patterns entsprechen im wesentlichen den Deklarationen, wie sie üblicherweise in Prüfmodulen im Abschnitt «Patterns» vorzunehmen sind (vgl. Abbildung 4). Im Unterschied dazu werden in 'CompoundPatterns.conf' jedoch ausschließlich Patterns, nicht jedoch zugeordnete Pattern Handler spezifiziert. Die Benennung eines Handlers entfällt, weil sämtliche Patterns in 'CompoundPatterns.conf' von einem gemeinsamen Standard-Handler bearbeitet werden.

Anstelle eines Handlers spezifiziert der Anwender pro Pattern zwei verschiedene Einstufungen — <*crit1*> und <*crit2*>:

- <*crit1*> gibt an, wie das **Auftreten** des spezifizierten Patterns in der Router-Konfigurationsbeschreibung einzustufen ist.

- <*crit2*> gibt an, wie das **Fehlen** des spezifizierten Patterns in der Router-Konfigurationsbeschreibung einzustufen ist.

Die Einstufung besteht jeweils aus einem einzigen Zeichen. Mögliche Belegungen sind { o, i, c, w, a } für OKAY, INFO, CHECK, WARN bzw. ALERT sowie '-' für «irrelevant». Im Prüflauf durchmustert das Prüfmodul die Router-Konfiguration nach Textstellen, die dem spezifizierten Pattern entsprechen. Wird ein entspre-

Copyright © Fraunhofer IESE 2003

chendes Textfragment gefunden, so schreibt das Prüfmodul eine Meldung gemäß Einstufung <*crit1*>. Läßt sich in der gesamten Konfigurationsbeschreibung kein solches Textfragment nachweisen, so wird eine Meldung gemäß Einstufung <*crit2*> protokolliert. Im Falle der Vorgabe '-' für <*crit1*> oder <*crit2*> wird die Meldung jeweils unterdrückt. Der ausgegebene Meldungstext nimmt Bezug auf das Text-Pattern, kann im übrigen aber vom Anwender nicht beeinflußt werden, sondern richtet sich nach den Vorgaben <*crit1*> und <*crit2*> sowie dem Maße, in dem ein Pattern erfüllt ist.

Das allgemeine Format einer Pattern-Definition lautet wie folgt:

```
<crit1> [<crit2>]   <pattern>
```

Die folgenden Beispiele illustrieren typische Anwendungsfälle:

```
aw      ip source-route
a-      ip route '0.0.0.0' '0.0.0.0' Ethernet ...
oc      version PROPER_VERSION
a-      version BAD_VERSION
```

Die erste Pattern-Definition besagt, daß das Auftreten der Anweisung 'ip source-route' in der Routerkonfiguration als kritisch zu werten ist und eine ALARM-Meldung nach sich ziehen soll. Auch das Fehlen einer solchen Anweisung gilt als Mangel und soll eine WARN-Meldung auslösen (erwünscht wäre: "no ip source-route" — oben nicht als Regel dargestellt).

Die zweite Pattern-Definition besagt, daß eine Routeranweisung, die mit 'ip route 0.0.0.0 0.0.0.0 Ethernet' beginnt, hochgradig unerwünscht ist und eine ALERT-Meldung hervorrufen soll. Fehlt ein solcher Eintrag in der Routerkonfiguration, so ist dies irrelevant und soll nicht protokolliert werden. (Der Bindestrich als zweites Zeichen in 'a-' darf auch weggelassen werden.)

Die Pattern-Definitionen drei und vier besagen zusammen, daß in der Konfigurationsbeschreibung die Angabe einer zulässigen Versionsnummer erwartet wird. Ist dies der Fall, so wird eine OKAY-Meldung protokolliert. Fehlt eine entsprechende Direktive in der Routerkonfiguration oder ist sogar eine unerwünschte Versionsnummer nachweisbar, so soll eine CHECK- bzw. ALERT-Meldung erfolgen.

Bei der Einstufung von Pattern genügt die Angabe von <*crit1*>. Die zweite Einstufung wird als '-' ergänzt, sofern nicht explizit etwas anderes angegeben ist. Als Patterns können elementare oder zusammengesetzte Patterns verwendet werden.

Copyright © Fraunhofer IESE 2003

Elementare Patterns

Ein elementares Pattern ist eine Textmuster-Beschreibung, die sich auf eine einzelne Konfigurationsklausel — eine isolierte Zeile der IOS Konfigurationsbeschreibung — bezieht. Patterns werden in Backus-Naur-Form spezifiziert. Das genaue Format und die möglichen Konstruktionselemente (z.B. Sequenz, Auswahl, Wiederholung, Negation) sowie die verfügbaren Grundmuster (Atoms) sind in Kapitel 3.2 im Abschnitt «Patterns» näher beschrieben. Dort finden sich auch Beispiele für typische Pattern-Spezifikationen.

Achtung: *Ein ungeschickt fomuliertes Pattern kann mehrdeutige Interpretationen zulassen. Dies wird in Kapitel 3.2 im Abschnitt «Konflikte: Gierige oder minmale Pattern-Interpretation?» genauer erläutert. Wird eine mehrdeutige Pattern-Interpretation erkannt, so unterbricht* CROCODILE *den Prüflauf mit einer Fehlermeldung. Der Anwender muß sein Pattern dann so verfeinern, daß die angezeigte Mehrdeutigkeit beseitigt wird.*

Zusammengesetzte Patterns

Ein zusammengesetztes Pattern (Compound Pattern oder kurz: Compound) entsteht durch Verknüpfung mehrerer elementarer Patterns. Die geschachtelte Verknüpfung mehrerer (Sub-)Compounds liefert ebenfalls ein gültiges Compound Pattern. Compounds dienen dazu, Textmuster zu spezifizieren, die mehrere Konfigurationsklauseln umfassen.

Zur Verknüpfung sind folgende Operatoren verfügbar:

- UND-Verknüpfung — *<compound1>* & *<compound2>*
- ODER-Verknüpfung — *<compound1>* | *<compound2>*
- XOR-Verknüpfung — XOR(*<compound1>*, …, *<compound n>)*
- Negation — NOT *<compound>*
- Bedingtes Pattern — IF (*<compound>*) [THEN] { *<then-compound>* }
[ELSIF (*<compound2>*) { *<elsif-compound>* }]*
[ELSE { *<else-compound>* }]
- Zähloperator — Count(<basic pattern>, <minoccur>,<maxoccur>)

UND-, ODER- bzw. XOR-Verknüpfung haben die offensichtliche Bedeutung: Das Compound gilt genau dann als aufgetreten, wenn beide, wenigstens einer bzw. genau einer der Operanden aufgetreten ist. Ein mit NOT negiertes Compound

Copyright © Fraunhofer IESE 2003

gilt genau dann als aufgetreten, wenn das Compound selbst nicht aufgetreten ist.

Ein bedingtes Compound gilt als aufgetreten, wenn das zutreffende THEN-, ELSIF- oder ELSE-Compound aufgetreten ist. Welches Compound das zutreffende ist, entscheidet sich danach, ob das IF-Compound oder eines der ELSIF-Compounds aufgetreten ist: Das erste nachgewiesene Bedingungs-Compound bestimmt den relevanten Zweig. Ist keines der Bedingungs-Compounds aufgetreten, ist das ELSE-Compound das relevante — sofern angegeben.

Der Zähloperator Count(…) besagt, daß das zugeordnete (elementare!) Pattern nur dann als aufgetreten gilt, wenn es wenigstens <*minoccur*> und höchstens <*maxoccur*> Mal nachgewiesen wurde. (Die Angabe der oberen Schranke darf entfallen mit der Bedeutung «beliebig oft»). Als Argument des Zähloperators ist nur ein elementares Pattern zulässig, weil sich die Häufigkeit beliebiger Compound Patterns im allgemeinen nicht sinnvoll definieren läßt.

Tabelle 2 zeigt eine förmliche Beschreibung der Compound-Pattern-Syntax.

Kontexte

Die Konfigurationsbeschreibungssprache des IOS unterscheidet verschiedene Modi. Einige Konfigurationsanweisungen sind nur möglich, wenn sich der Router in einem bestimmten Betriebssystem-Modus befindet. Um solche Anweisungen zu erteilen, muß zuvor ein Modus-Wechsel veranlaßt werden. Nachfolgende Anweisungen sind dann nur im Kontext des eingestellten Modus wirksam, wie auf Seite 34 als «Klauselkontext» eingeführt.

Klauselkontexte eines bestimmten Typs können in einer IOS Konfigurationsbeschreibung in der Regel in mehreren Ausprägungen (Instanzen) auftreten. Ein Beispiel ist etwa der Interface-Modus zum Konfigurieren von Router-Schnittstellen. Da ein Router mehrere Schnittstellen hat, beziehen sich alle Interface-Konfigurationsanweisungen auf die Schnittstelle, die zuvor beim Moduswechsel als jeweilige Instanz benannt wurde. Die gleiche Anweisung kann sich daher in der Konfigurationsbeschreibung mehrfach wiederholen, z.B. je einmal für jedes vorhandene Interface.

Um diesem Umstand Rechnung zu tragen, genügt es bei Pattern-Definitionen im allgemeinen nicht, ein Textmuster anzugeben. Vielmehr muß auch unterschieden werden, in welchem Kontext und in welcher Kontextinstanz das Pattern geprüft werden soll. So ist etwa das Auftreten der Klausel 'shutdown' nur dann sinnvoll interpretierbar, wenn klar ist, auf welche Schnittstelle sich der Befehl bezieht.

Copyright © Fraunhofer IESE 2003

Compound	::=	*Term* [**|** *Term*]* | **if (** *Compound* **)** [**then**] **{** *Compound* **}** [**elsif (** *Compound* **)** [**then**] **{** *Compound* **}**]* [**else {** *Compound* **}**]
Term	::=	*Factor* [**&** *Factor*]*
Factor	::=	[**NOT**] { *Pattern* | *Counted_Pattern* | [**Global**] **(** *Compound* **)** | **XOR (** *Compound* { **,** *Compound* }* **)** }
Counted_Pattern	::=	**Count (** *Pattern***,** <min> [**,** <max>] **)**
Pattern	::=	{ [**!**] { *Atom* | *Nonatom* | *Macro* } }*
Nonatom	::=	**{** *Pattern* [**|** *Pattern*] **}** [*****] | **[** *Pattern* [**|** *Pattern*] **]** [*****]
Atom	::=	<IOS Keyword> | <numeral> | <min>**-**<max> | **'**<string>**'** | **STRING** | **DSTRINGD** | **WORD** | **NAME** | **NUM** | **IPADDR** | **PATTERN** | **...** | **/**<Perl regular expression>**/**
Macro	::=	<macro name> **::=** *Pattern*

Tabelle 2 Syntax zur Bildung von Compound Patterns. (Zeichen in Fettdruck gehören zur Syntax, alle andere Zeichen sind Metazeichen zur Syntaxbeschreibung, die selbst nicht zum Pattern gehören.)

Daher wurde beim Entwurf des Prüfmoduls 'CompoundPatterns' das Konzept der sogenannten Kontext-Definition ersonnen, um Patterns auf bestimmte Klauselkontexte einzuschränken. Dabei ist das im folgenden beschriebene Kontext-Konzept sogar noch allgemeiner, da es nicht unbedingt einem IOS-

Copyright © Fraunhofer IESE 2003

Konfigurationsmodus entsprechen muß; in der Praxis ist die Darstellung von IOS-Modi jedoch der wesentliche Verwendungszweck.

Das allgemeine Format einer Kontextdefinition lautet wie folgt:

```
<begin (basic) pattern> {
  …
} <end (basic) pattern>
```

Man beachte, daß *<begin pattern>* und *<end pattern>* je eine eigene Zeile beanspruchen und nicht in eine gemeinsame Zeile geschrieben werden dürfen. Es muß sich jeweils um elementare Pattern handeln, Compounds sind nicht zulässig und wären an dieser Stelle auch unsinnig. Zwischen den geschweiften Klammern können beliebig viele Compound-Definitionen eingetragen werden, jede in einer neuen Zeile beginnend. Diese werden vom Prüfmodul 'Compound-Patterns' nur im angegebenen Kontext ausgewertet, außerhalb des Kontexts aber ignoriert.

Das *<begin pattern>* ist ein beliebiges elementares Pattern. Es beschreibt die Eröffnungsklausel, mit der unter IOS der gewünschte Klauselkontext eingeleitet wird. Sobald das angegebene Textmuster in der Router-Konfigurationsbeschreibung auftritt, wird der Kontext aktiviert: Bis auf weiteres wertet nun das Prüfmodul die innerhalb der geschweiften Klammern stehenden Compound-Definitionen für die vorliegende Kontextinstanz aus. Es verarbeitet sie wie gewöhnliche, nicht kontextgebundene Compounds. Beginnend bei der auf *<begin pattern>* folgenden Klausel wird die Router-Konfigurationsbeschreibung Zeile für Zeile analysiert.

Dies setzt sich so lange fort, bis das *<end pattern>* in der Konfigurationsbeschreibung auftritt, ebenfalls ein elementares Pattern. Wird ein entsprechendes Textmuster erkannt, so schließt das Prüfmodul den zugeordneten Kontext und deaktiviert alle im Kontext definierten Compound-Definitionen. Danach ist der Kontext inaktiv, bis in der Konfigurationsbeschreibung erneut ein Textmuster auftritt, das dem *<begin pattern>* entspricht.

Aktivierung und Deaktivierung eines Kontexts können sich beliebig oft wiederholen. Die in geschweiften Klammern eingeschlossenen Compounds werden dabei separat für jede neue Kontextinstanz ausgewertet. Beliebig viele verschiedene Kontexte können gleichzeitig oder sogar zeitlich überlappend aktiv sein.

Das folgende Beispiel illustriert den Einsatz von Kontext-Definitionen:

```
interface Loopback ... {
  ow  if (NOT shutdown) { ip address IPADDR IPMASK }
} interface ...
```


Copyright © Fraunhofer IESE 2003

```
interface !Loopback ... {
  ow  if (NOT shutdown) { description ... &
                          no ip redirects &
                          no ip directed-broadcast &
                          no ip proxy-arp &
                          no cdp enable
                        }
} interface ...
```

Im vorliegenden Fall sollen Konfigurationsmerkmale verschiedener Interfaces überprüft werden, wobei zwischen sogenannten Loopback-Interfaces und sonstigen Interface-Typen unterschieden wird.

Die erste Kontext-Definition macht Prüfvorgaben für alle Loopback-Interfaces. Sobald eine Konfigurationsanweisung erkannt wird, die mit den Schlüsselwörtern 'interface Loopback' beginnt, wird die Compound-Definition für Loopback-Interfaces vom Prüfmodul aktiviert. (Die drei Punkte '...' stehen wörtlich in der Compound-Spezifikation. Sie bilden ein atomares Pattern mit der Bedeutung «beliebige Zeilenfortsetzung».) Im Beispiel wird überprüft, ob das Interface abgeschaltet ist. Ist dies der Fall, so ist die bedingte Regel nicht anwendbar und insoweit erfüllt (→ OKAY-Meldung). Falls die IF-Bdingung zutrifft und das Interface nicht abgeschaltet ist, so besagt die Regel, daß das Loopback-Interface eine IP-Adresse tragen muß (→ OKAY). Andernfalls erscheint eine WARN-Meldung im Protokoll.

Der Kontext der Loopback-Konfiguration endet, sobald in der Router-Konfiguration eine neue Interface-Anweisung ('interface ...') — gleich welchen Typs — erkannt wird. Sollte es sich zufällig erneut um ein Anweisung der Form 'interface Loopback …' handeln, so beschließt dies den Kontext, der unmittelbar darauf wieder für eine neue Interface-Instanz eröffnet wird.

Die zweite Kontext-Definition macht Prüfvorgaben für alle sonstigen Interfaces außer Loopback-Interfaces. Der Beginn des Kontexts ist an Konfigurationsanweisungen zu erkennen, die mit 'interface' beginnen und sich *nicht* mit 'Loopback' fortsetzen. Auch dieser Kontext endet, sobald ein neues 'interface ...' in der Konfigurationsbeschreibung auftritt. Die Klausel im Inneren der Kontextdefinition besagt im wesentlichen, daß für jedes Interface, das nicht ausdrücklich abgeschaltet ist, die folgenden Merkmale vereinbart sein sollen: 'description …', 'no ip redirects', 'no ip directed-broadcast', 'no ip proxy-arp' sowie 'no cdp enable'.

Globaler Kontext versus Klauselkontext

Alle IOS-Klauseln, die nicht in einem gesonderten Klauselkontext auftreten, werden dem sogenannten Globalen Kontext zugerechnet und als «globale Klauseln» bezeichnet. Globale Klauseln beeinflussen die Wirkung von Kontext-

Copyright © Fraunhofer IESE 2003

klauseln. Daher ist es oft erforderlich, innerhalb eines kontextabhängigen Compounds auch globale Sub-Compounds zu berücksichtigen. Der Operator 'Global(…)' ermöglicht es, innerhalb eines Kontexts auf ein globales Compound Bezug zu nehmen. Ein typisches Beispiel ist etwa:

```
interface Serial ... {
  ow   Global(no cdp run) | no cdp enable
} interface ...
```

Diese Regel besagt, daß das «Cisco Discovery Protocol» zur automatischen Erkennung erreichbarer Schnittstellen entweder auf globaler Ebene — d.h. für alle Interfaces gemeinsam — deaktiviert sein soll, oder gezielt für alle Serial Interfaces im Interface-Kontext abzuschalten ist.

Die Verwendung des Global-Operators außerhalb eines Kontext-Spezifikation hat keine Wirkung auf das eingeschlossene Compound, ist aber zulässig.

Befundausgabe

Der Befund zu jedem Compound Pattern wird als Annotation ausgegeben. Abbildung 12 zeigt einen typischen Ausschnitt einer solchen Ausgabe.

Je nachdem, welche Schwere <*crit*> der Befund hat und abhängig davon, ob es sich um ein nachgewiesenes oder fehlendes Pattern handelt, variiert die Formulierung der Befundmeldung, auf die der Anwender keinen Einfluß nehmen kann. In der Regel werden nur die Pattern-Komponenten genannt, die zum Erfolg bzw. zum Fehlschlag der Prüfung beigetragen haben. Terme innerhalb des Compound

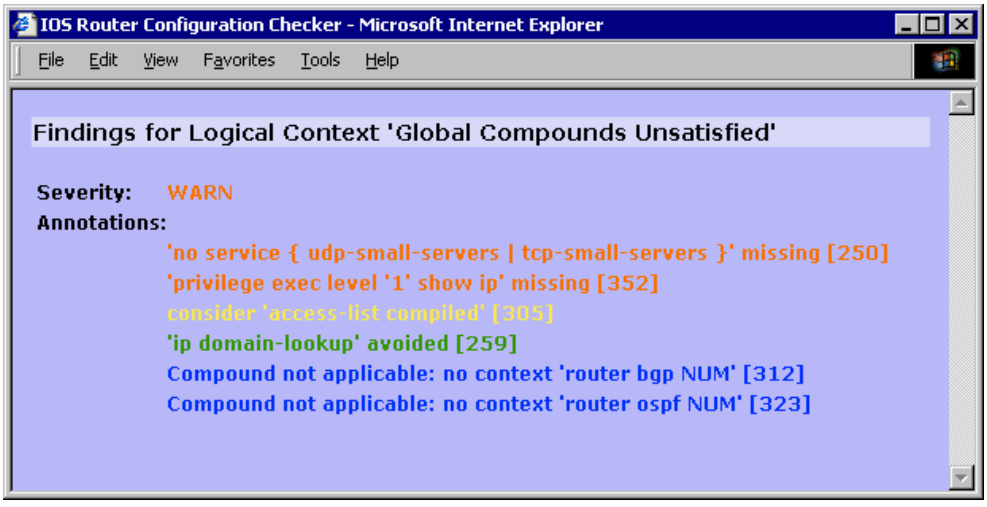

Abbildung 12 Typische Befundausgaben des Moduls 'CompoundPatterns'

Copyright © Fraunhofer IESE 2003

Patterns, die keinen Einfluß auf das Gesamtergebnis hatten (zum Beispiel nicht erfüllte Teilterme einer insgesamt erfüllten ODER-Bedingung) werden in der Ausgabe der Übersichtlichkeit halber unterdrückt.

Der im Befund genannte Ausdruck kann daher im Format deutlich von dem zugrunde liegenden Compound abweichen. Um dennoch eine eindeutige Zuordnung zu gewährleisten, wird am Ende der Befundmeldung in eckigen Klammern die Zeilennummer der Konfigurationsdatei 'CompoundPatterns.conf' ausgewiesen, in der das Compound Pattern deklariert ist. Dort sollte im Bedarfsfall ein Kommentar plaziert werden, der die Bedeutung jeder nicht-offensichtlichen Prüfregel ausführlich erklärt.

Bemerkungen

Das Prüfmodul 'CompoundPatterns' deckt einen beachtlichen Einsatzbereich ab. Zu den typischen Prüfaufgaben, die von 'CompoundPatterns' übernommen werden können, zählen zum Beispiel:

- Prüfung, ob geforderte Router-Dienste und Betriebsmodi ausdrücklich aktiviert und unerwünschte Dienste ausdrücklich abgeschaltet sind
- Prüfung, ob bestimmten Kommandos das vorgeschriebene Privilege Level zugewiesen ist
- Prüfung, ob Banner-Texte vereinbart sind und bei passender Gelegenheit angezeigt werden
- Prüfung, ob Paßwörter vereinbart und zugewiesen sind, und ob es sich um sicher codierte Paßwörter handelt.

Trotz seiner Vielseitigkeit ist das Prüfmodul einfach zu bedienen. Der Anwender muß keine eigenen Prüfmodule programmieren, um Prüfvorgaben nach seinen Wünschen zu spezifizieren. Das BNF-artige Format der Compound-Spezifikationen und die Verknüfungsmöglichkeiten mit logischen Operatoren ermöglicht sehr vielschichtige Definitionen, insbesondere auch bedingte Prüfregeln.

Das Modul kann aufgrund seiner einfachen Konfigurierbarkeit auch als Trainingsmöglichkeit für solche Anwender angesehen werden, die eigene Prüfmodule entwerfen möchten, zuvor aber Erfahrungen mit der Pattern- und Makro-Syntax sammeln wollen. Pattern-Spezifikationen können mittels 'CompoundPatterns' ohne großen Aufwand ausprobiert werden, ehe sie als Trigger für Pattern Handler in einem neuen Prüfmodul Verwendung finden.

Copyright © Fraunhofer IESE 2003

Der Preis für die Vielseitigkeit des Prüfmoduls ist sein eingeschränkter Reaktionsspielraum beim Erkennen eines Patterns. Der Anwender hat keine Möglichkeit, eigene Meldungstexte vorzugeben, um das Prüfergebnis genau zu beschreiben.[8] Statt dessen wird nur das Fehlen/Vorhandensein gewisser Textmuster gemeldet und gegebenenfalls beanstandet. Der Meldungswortlaut ist standardisiert und daher für Anwender ohne tiefe Router-Kenntnisse mitunter schwer verständlich.

Ein verbleibender Mangel der Compound-Spezifikationen ist das Fehlen von Möglichkeiten, Instanzennamen und Pattern-Bestandteile beim Auftreten an Variablen zu binden, um im weiteren Verlauf des Compounds darauf Bezug zu nehmen. Mit einem solchen Mechanismus wäre es zum Beispiel möglich, Regeln folgenden Typs zu formulieren:

```
# Wenn ein Line-Interface durch eine Access-Control-Liste geschützt
# wird, so muß diese ACL an anderer Stelle auch definiert werden

line VTY ... {
  oa  if (access-class ACL_NUMBER) {
          Global(access-list ACL_NUMBER ...)
      }
} line ...
```

Obwohl eine entsprechende Erweiterung des Compound-Mechanismus zunächst naheliegend erscheint, so hat sie doch erhebliche Folgewirkungen auf den Compound-Kalkül, die uns bisher vor einer Realisierung zurückschrecken lassen. Eine Konsequenz ist etwa, daß man bei Variablenbindung das Auftreten mehrerer Instanzen berücksichtigen muß. Die Bindung ist daher potentiell mehrdeutig, was schließlich dazu führt, daß man Quantoren (Existenz-Quantor und All-Quantor) einführen muß, um sinnvolle Compound-Interpretationen zu erhalten. Dies ist nicht nur technisch sehr aufwendig und in der Notation unhandlich — es ist dem Normalanwender auch kaum noch zumutbar.

4.3 Modul 'IngressEgress'

Router, die an der Grenze eines Netzwerks positioniert sind und die Verbindung zu anderen Netzwerken herstellen (sogenannte Perimeter Router), sollten an der Übergangsschnittstelle eingehende und ausgehende Nachrichten auf Plausibilität prüfen. In eingehender Richtung (ingress) schützen sie damit das eigene Netz gegen unliebsame, fehlerhafte oder gar manipulierte Datenpakete; in ausgehender Richtung (egress) verhindern sie, daß vom eigenen Netz Störungen oder Manipulationsversuche in Richtung anderer Netze ausgehen.

8 Technisch wäre es ohne weiteres möglich, die Pattern-Definitionen um Meldungstexte anzureichern. Dies brächte aber zunehmende Lasten bei der Erstellung der Konfigurationsdatei 'CompoundPatterns.conf' mit sich. Daher sind dem Ausdrucksvermögen der Konfigurationssyntax recht enge Grenzen gesetzt.

Copyright © Fraunhofer IESE 2003

IOS-Router realisieren die sogenannte Ingress- und Egress-Filterung üblicherweise mit Hilfe geeigneter Access-Listen. Das Prüfmodul 'IngressEgress' ermöglicht es, genaue Vorgaben für die Filterwirkung solcher Access-Listen zu machen und deren Einhaltung zu überprüfen.

Das Prüfmodul 'IngressEgress' baut auf Analysen des Prüfmoduls 'Connectivity' auf (siehe Kapitel 4.4, Seite 64 ff) und kann daher nur im Verbund mit diesem aktiviert werden. Ist 'Connectivity' für einen Prüflauf nicht aktiviert, so ist 'IngressEgress' nicht lauffähig. Das Prüfwerkzeug kontrolliert diese Abhängigkeit automatisch und lädt das Partnermodul 'Connectivity' notfalls automatisch nach.

Prüfvorgaben

Die Prüfvorgaben des Moduls 'IngressEgress' sind frei konfigurierbar. Den konkreten Prüfauftrag erhält das Modul über die Modul-Konfigurationsdatei 'IngressEgress.conf' im Unterverzeichnis 'Configuration' (vgl. Abbildung 10, Seite 43).

Die Konfigurationsdatei enthält Anforderungen an Ingress/Egress-Filterung für

- Interfaces,
- Line Interfaces oder
- Access-Listen.

Dazu kann der Anwender sowohl Whitesets als auch Blacksets vorgeben. *Whitesets* geben an, welche Nachrichtentypen von einem Interface, einem Line Interface oder einer Access-Liste *mindestens* akzeptiert werden müssen. *Blacksets* legen fest, welche Nachrichtentypen *mindestens* abgewiesen werden sollen. Findet sich der Typ einer Nachrichten weder in einem Whiteset noch in einem Blackset, so kann die entsprechende Nachricht aufgrund der Router-Konfiguration akzeptiert oder abgewiesen werden — das Prüfmodul läßt beides unbeanstandet.

Prinzip der Filterspezifikation

Syntax und Semantik, um Anforderungen an Ingress/Egress-Filterung vorzugeben, sind eng an IOS-Befehle angelehnt. Dies bietet dem Anwender ein vertrautes, leicht zu beherrschendes Befehlsformat.

Das Beschreibungsprinzip ist einfach: Der Anwender schreibt eine Spezifikation in Form einer «Filter-Liste», die der gewünschten Filterwirkung entspricht. Das Prüfmodul ermittelt daraus ein Whiteset (Menge aller zulässigen Nachrichten)

Copyright © Fraunhofer IESE 2003

und ein Blackset (Menge aller unzulässigen Nachrichten). Später im Prüflauf wird dann ermittelt, ob alle Elemente des Whitesets von der angegebenen Access-Liste oder dem Interface bzw. Line Interface akzeptiert werden, und ob alle Elemente des Blacksets tatsächlich zurückgewiesen werden. Nicht akzeptierte Whiteset-Anteile oder fälschlich akzeptierte Blackset-Anteile werden jeweils in Form einer Warnung aufgelistet (mehr dazu in Abbildung 14 auf Seite 60).

Abbildung 13 zeigt eine Beispielspezifikation und das Prinzip, nach dem daraus Whiteset und Blackset ermittelt werden. Eine Spezifikation besteht aus einer Folge von 'deny'- und 'permit'-Klauseln. Sie kommen der Reihe nach, von oben nach unten, zur Anwendung. Handelt es sich um eine 'deny'-Klausel, so wird aus der Menge aller möglichen Nachrichten («ip any any») die Teilmenge mit den spezifizierten Attributen abgezweigt und dem Blackset zugeordnet. Gemäß Abbildung 13 werden zum Beispiel alle ICMP-Nachrichten dem Blackset zugeordnet.

Auf die Menge der verbleibenden Nachrichten wird die jeweils nächste Klausel angewendet, in Abbildung 13 ein 'permit'. Jede 'permit'-Klausel zweigt die entsprechende Teilmenge an Nachrichten in das Whiteset ab. Im Beispiel sind dies alle TCP-Nachrichten an die IP-Adresse 135.234.100.30 mit Zielport 80. Alle übrigen Nachrichten — im Beispiel alle außer ICMP-Nachrichten und TCP-Nachrichten an den bezeichneten Zielport — bleiben übrig und werden an die

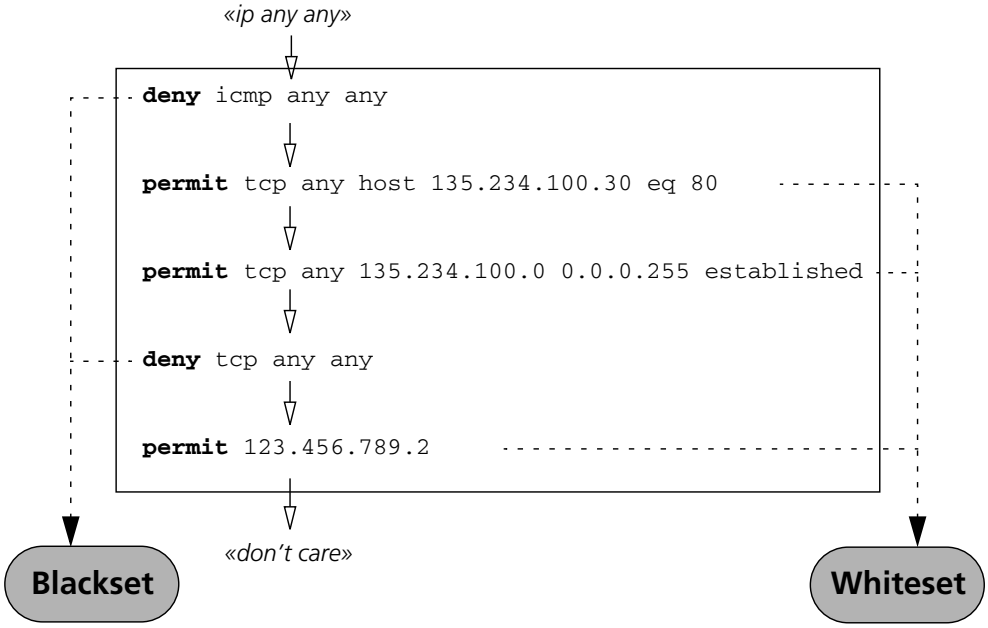

Abbildung 13 Spezifizieren von Ingress/Egress-Vorgaben: Whiteset und Blackset

Copyright © Fraunhofer IESE 2003

nächste Filterklausel weitergereicht. Auf diese Weise wird Klausel für Klausel abgearbeitet.

Nach dieser Methode erhält man eine reine Whiteset-Spezifikation, indem man ausschließlich 'permit'-Klauseln verwendet. Wird der Filter ausschließlich mit 'deny'-Klauseln beschrieben, ergibt sich eine reine Blackset-Vorgabe. Mischformen wie in Abbildung 13 sind ebenso zulässig.

Die Interpretation der Filterspezifikationen entspricht demnach im wesentlichen der Methode, nach der IOS-Router Access-Listen auswerten. Der Anwender sollte damit gut vertraut sein. Im Unterschied zu Access-Listen wird am Ende einer 'IngressEgress' Filterspezifikation jedoch kein implizites «deny any» angewendet. Nachrichten, die nach der letzten Klausel übrig geblieben sind und weder dem Whiteset noch dem Blackset zugeordnet wurden, bleiben statt dessen unreglementiert: Sie können vom Router nach Belieben akzeptiert oder zurückgewiesen werden. Bei einer Access-Liste werden dagegen alle übrig gebliebenen Nachrichten automatisch vom Router verworfen. Soll mit einer Filterspezifikation exakt die Semantik einer Access-Liste nachgebildet werden, so sollte als letzte Klausel einfach «deny any» an die Spezifikation angefügt werden.

Das Verfahren liefert schließlich zwei Mengen: Das Whiteset enthält (konzeptionell) alle denkbaren Nachrichten, die aufgrund der Filterspezifikation zulässig sein sollen; das Blackset umfaßt (konzeptionell) alle denkbaren Nachrichten, die laut Filterspezifikation unzulässig sein sollen. Das Modul 'IngressEgress' ist nun in der Lage, für jede einzelne dieser Nachrichten zu überprüfen, ob diese aufgrund der Router-Konfiguration im Betrieb tatsächlich akzeptiert bzw. verworfen wird.[9] Jede Abweichung wird genau protokolliert.

Abbildung 14 zeigt einen typischen Ausschnitt aus dem Ergebnis einer 'IngressEgress'-Prüfung. In diesem Beispiel wird eine Whiteset- sowie eine Blackset-Vorgabe für den Zugriff auf Line Interfaces der Form "vty …" überprüft, und zwar in Eingaberichtung ("in"). Die Abbildung zeigt in der oberen Hälfte die Vorgabe, darunter das Prüfergebnis für das Line Interface 'vty 0', das gemäß Router-Konfiguration durch die Standard-Access-Liste Nr. 2 geschützt ist. Man erkennt, daß die Access-Liste 2 bezüglich des 'telnet' Ports offenbar zu restriktiv, bezüglich gewisser Nachrichtenursprünge jedoch zu durchlässig ist.

Die oben skizzierte Spezifikationsmethode funktioniert unabhängig von der Reihenfolge oder der Form der Filterklauseln: Maßgeblich ist allein die Wirkung, die sich aus dem Zusammenspiel aller Klauseln ergibt. Das bedeutet insbe-

9 Die Beschreibung bezieht sich auf ein logisches Konzept: In der technischen Durchführung wird natürlich darauf verzichtet, die beschriebenen Mengen tatsächlich — Element für Element — zu bilden. Eine solche naive Vorgehensweise würde an der enormen Kardinalität der Mengen scheitern. Statt dessen werden Whitesets und Blacksets nur symbolisch beschrieben und mittels Umformungsschritten auf der Symbolebene analysiert.

Copyright © Fraunhofer IESE 2003

```
Ingress/Egress for Line (in):
-------------------------------------------------

PATTERN = 'vty ...'
Whiteset = {
tcp 153.96.133.*  *.*.*.* eq telnet
}
Blackset = {
ip [0-152,154-255].*.*.*  *.*.*.*
ip 153.[0-95,97-255].*.*  *.*.*.*
ip 153.96.[0-132,134-255].*  *.*.*.*
[0-5,7-255] 153.96.133.*  *.*.*.*
tcp 153.96.133.*  *.*.*.* [0-22,24-65535]
}

Line 'vty 0'
Accesslist: 2
--> Whiteset only partially supported. Missing:
tcp 153.96.133.*  *.*.*.* eq telnet
--> Blackset only partially supported. Missing:
ip 151.200.19.52  *.*.*.*
ip 164.47.133.2  *.*.*.*
ip 217.31.34.129  *.*.*.*
```

Abbildung 14 Typisches Beispiel für das Ergebnis einer 'IngressEgress'-Prüfung

sondere, daß sich die vorgegebenen Filterklauseln nicht unbedingt wörtlich in der Router-Konfiguration wiederfinden müssen. Es genügt, daß die Router-Konfiguration insgesamt den geforderten Effekt erzielt, nämlich, das Whiteset zu akzeptieren und das Blackset zu verwerfen — mit welchen 'access-list'-Konstrukten auch immer! Dies unterscheidet das Prüfmodul 'IngressEgress' deutlich von anderen, im Internet kursierenden einfachen Prüfskripten, die Vorgabe und Konfigurationsbeschreibung lediglich auf wörtliche Übereinstimmung prüfen.

Verfügbare 'permit'- und 'deny'-Klauseln

Filterklauseln orientieren sich am Format der Standard Access-Listen sowie der Erweiterten Access-Listen. Die Syntax für gültige Filterklauseln lautet (im Standardformat)

```
{permit|deny} {any | IPADDR [WILDCARDMASK]}
```

oder (im erweiterten Format)

```
{permit|deny} PROTOCOL SUBNET [PORTSPEC] SUBNET [PORTSPEC|ICMPSPEC]
[established]
```


Copyright © Fraunhofer IESE 2003

wobei:

```
PROTOCOL ::= {0-255 | <Protokollname>}
SUBNET   ::= {IPADDR WILDCARDMASK | any | host IPADDR}
PORTSPEC ::= {{lt|gt|eq|neq} <Port> | range <Port> <Port>}
ICMPSPEC ::= {0-255 [0-255] | <Typname>}
```

Dies entspricht der Parametrisierung und Wirkung entsprechender IOS-Befehle.

Anders als bei IOS Access-Listen können Standardklauseln beliebig mit erweiterten Klauseln gemischt werden. Auch können Vorgaben im Standardformat erstellt werden, um Erweiterte Access-Listen zu prüfen — und umgekehrt.

Wie eingangs erwähnt, können Filtervorgaben gezielt bestimmten Access-Listen oder Interfaces zugeordnet werden. Damit lassen sich für die verschiedenen Router-Zugänge unterschiedliche Filtervorgaben festlegen.

Prüfvorgaben für Access-Listen

Im einfachsten Fall gibt der Anwender Kriterien für die Filterwirkung von Access-Listen vor. Es liegt dann in der Verantwortung des Nutzers, diese Access-Listen entsprechend ihrer Bestimmung einzelnen Schnittstellen zuzuordnen.

Das Spezifikationsformat für 'access-list'-Prüfvorgaben lautet wie folgt:

```
access-list <Pattern> {
 <Liste von 'permit' und 'deny' Filterklauseln>
}
```

Das Pattern bezeichnet eine BNF-artige Syntaxbeschreibung (entsprechend dem Pattern-Format für Pattern Handler, vgl. Kapitel 3.2) und legt fest, welche Access-Listen von der Vorgabe betroffen sein sollen. Nur solche Listen, deren Name dem Pattern entspricht, fallen unter die Vorgabe. Wird als Pattern zum Beispiel "[1-99]" eingesetzt, so bezieht sich die Vorgabe auf alle Standard Access-Listen; das Pattern "..." bedeutet hingegen, daß die Vorgabe für beliebige Access-Listen gelten soll. Beliebig komplexe Patterns sind spezifizierbar.

Prüfvorgaben für Interfaces

Der Anwender kann eine Filtervorgabe auch gezielt bestimmten Interfaces zuordnen. Das Prüfmodul analysiert dann diese Schnittstellen, unabhängig davon, durch welche Access-Listen sie geschützt werden.

Das Spezifikationsformat dafür lautet ähnlich wie oben, allerdings muß zusätzlich angegeben werden, ob sich die Vorgaben auf ankommende ("in"), abgehende

Copyright © Fraunhofer IESE 2003

```
! RFC1918 inbound address filtering

interface { Ethernet | Serial } ... in {
    deny   ip 10.0.0.0 0.255.255.255 any
    deny   ip 127.0.0.0 0.255.255.255 any
    deny   ip 172.16.0.0 0.15.255.255 any
    deny   ip 192.168.0.0 0.0.255.255 any
}

!  RFC1918 outbound address filtering

interface { Ethernet | Serial } ... out {
    deny   ip any 10.0.0.0 0.255.255.255
    deny   ip any 127.0.0.0 0.255.255.255
    deny   ip any 172.16.0.0 0.15.255.255
    deny   ip any 192.168.0.0 0.0.255.255
}
```

Abbildung 15 Beispiel für eine 'IngressEgress'-Prüfvorgabe für Interfaces

("out") oder auf alle Nachrichten in beiden Kommunikationsrichtungen ("inout") beziehen sollen:

```
interface <Pattern> {in|out|inout} {
 <Liste von 'permit' und 'deny' Filterklauseln>
}
```

Abbildung 15 zeigt ein Beispiel für eine Interface-Vorgabe, die sich an den Empfehlungen zur Filterung nichtöffentlicher IP-Adreßbereiche (laut [RFC1918]) orientiert. Die Vorgabe soll im Beispiel für alle Interfaces vom Typ 'Serial' und 'Ethernet' gelten. Zeilen der Filterspezifikation, die mit '!' bzw. '#' beginnen, gelten — wie traditionell unter IOS bzw. Unix — als Kommentare und werden vom Prüfwerkzeug ignoriert.

Prüfvorgaben für Line Interfaces

Line-Interfaces können genau wie Interfaces mit Filtervorgaben belegt werden. Das Spezifikationsformat ist entsprechend:

```
line <Pattern> {in|out|inout} {
 <Liste von 'permit' und 'deny' Filterklauseln>
}
```


Copyright © Fraunhofer IESE 2003

Verwendung von Makros

Bei der Formulierung von Filterlisten kann es wiederkehrende Textsequenzen geben oder Spezifikationsbausteine, die aus Gründen der Lesbarkeit und Wartbarkeit mit einem symbolischen Namen belegt werden sollten. Dazu bietet das Prüfmodul die Möglichkeit, in 'IngressEgress.conf' Makros zu deklarieren. Das Format dazu lautet:

<MACRO NAME> **::=** *<Pattern>*

Eine Makro-Deklaration darf nur eine Zeile umfassen, und der frei wählbare Makroname darf nur Großbuchstaben enthalten. Jedes beliebige BNF Pattern darf verwendet werden — soweit dies mit dem Verwendungszweck des Makros vereinbar ist. Abbildung 16 zeigt Beispiele für die Verwendung von Makros.

Bemerkungen

Der besondere «Charme» des 'IngressEgress'-Moduls liegt zum einen in seiner einfachen und flexiblen Konfigurierbarkeit, zum anderen in seinen — unseres Wissens einmaligen — Fähigkeiten, beliebig komplexe Whitesets und Blacksets auf rein logischer Ebene zu handhaben, ohne dabei auf die Art und Weise ihres Zustandekommens Bezug nehmen zu müssen: Auch wenn eine Filterspezifikation keinerlei formale Ähnlichkeit mit den relevanten Teilen der Konfigurationsbeschreibung hat, kann das Prüfmodul dennoch erkennen, ob beide Beschreibungen logisch äquivalent sind.

```
NETMASK   ::= 0.0.0.255
LOCALNETA ::= 147.56.103.0 NETMASK
LOCALNETB ::= 147.56.104.0 NETMASK
LOCALNETC ::= 147.56.105.0 NETMASK

OUTBOUND  ::= Serial [0-1]

interface OUTBOUND out {
    deny   ip any 10.0.0.0 0.255.255.255
    deny   ip any 127.0.0.0 0.255.255.255
    deny   ip any 172.16.0.0 0.15.255.255
    deny   ip any 192.168.0.0 0.0.255.255
    permit ip LOCALNETA any
    permit ip LOCALNETB any
    permit ip LOCALNETC any
}
```

Abbildung 16 Beispiele für die Verwendung von Makros in 'IngressEgress.conf'

Copyright © Fraunhofer IESE 2003

Auch aus technischer Sicht ist dieses Prüfmodul bemerkenswert. Es greift bei der Auswertung seiner Konfigurationsdatei 'IngressEgress.conf' auf die bereits vorhandene Funktionalität von Parser und Patternmatcher zurück. Dies ermöglicht es einerseits, Spezifikationsklauseln unterschiedlichster Ausprägung sicher zu erkennen, zum anderen können Tippfehler in der Spezifikation nahezu zeichengenau lokalisiert und präzise beanstandet werden. Der Rückgriff auf gewöhnliche BNF Pattern bei der Festlegung des Spezifikationsformats bietet bei Bedarf auch die Möglichkeit, das Syntaxformat einzelner Klauseln mit wenig Aufwand bedarfsgerecht zu modifizieren oder zu erweitern.

4.4 Modul 'Connectivity'

Das Prüfmodul 'Connectivity' dient dazu, die Verbindungstopologie des Routers und der ihn umgebenden Subnetze und Netzwerkknoten zu ergründen und in verständlicher Form darzustellen. Das Fernziel besteht darin, alle benötigten Informationen für eine Darstellung zu liefern, aus der die zulässigen Informationsflüsse zwischen den Router-Interfaces, Verbindungsrouten sowie potentiell beteiligte Kommunikationspartner, verwendbare Kommunikationsprotokolle und vorgesehene Filterregeln hervorgehen. Derzeit werden die erhobenen Daten nur in textueller Form protokolliert. Die graphische Umsetzung muß per Hand vorgenommen werden, was aber mit Hilfe des Prüfprotokolls leicht zu bewerkstelligen ist. Ein typisches Beispiel für die Art von Skizze, die auf diese Weise erstellt werden kann, liefert Abbildung 17.

Prüfvorgaben

Das Modul 'Connectivity' ist nicht parametrisierbar, besitzt also keine zugeordnete Konfigurationsdatei. Alle relevanten Pattern und die dazu gehörenden Pattern Handler sind im Modulcode fest vorgegeben.

Das Prüfmodul führt nur am Rande Sicherheitsanalysen im engeren Sinne durch. Hauptsächlich ermittelt es eine Übersicht über die Netzwerkanbindungen des Routers. Soweit dabei Merkwürdigkeiten und Ungereimtheiten zutage treten, werden diese selbstverständlich protokolliert. Der Schwerpunkt liegt allerdings auf Informationsbeschaffung und -aufbereitung, um spätere manuelle Analysen zu unterstützen. Dazu werden die nachfolgend genannten Daten erhoben.

Interfaces

Für alle IP-Schnittstellen werden die wesentlichen Daten zum Interface und zu den darüber erreichbaren Knoten und Subnetzen protokolliert, darunter:

- IP- und Subnetzadresse(n) der Schnittstelle sowie IP Broadcast-Adresse

Copyright © Fraunhofer IESE 2003

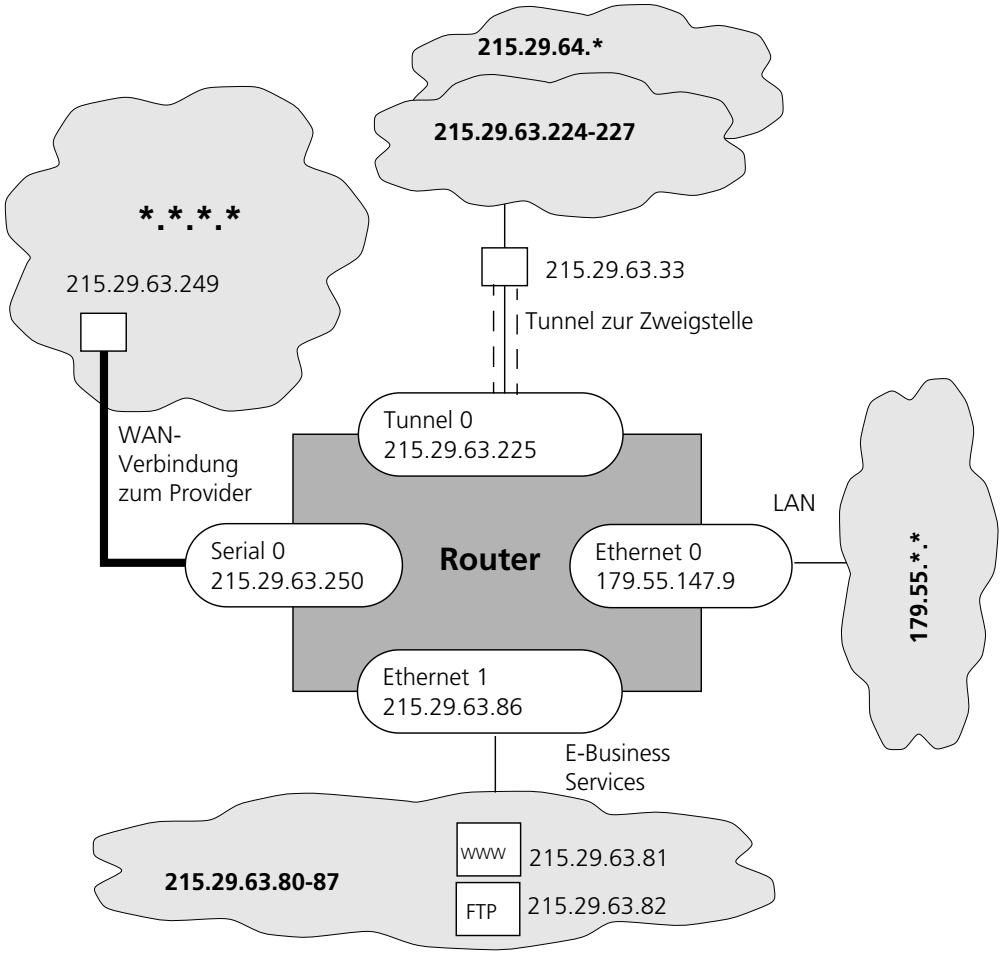

Abbildung 17 Beispiel für eine Verbindungstopologie, wie sie mittels 'Connectivity' rekonstruiert werden kann

- Klartextbeschreibung (soweit in der Konfigurationsbeschreibung enthalten),
- Aktivierungszustand,
- über das Interface erreichbare Subnetze und Knoten sowie Protokolle, mit denen diese Knoten ansprechbar sind (soweit aus der Konfigurationsbeschreibung ableitbar),
- vorgesehene Access-Liste für eingehenden Verkehr und resultierende akzeptierte Protokollverbindungen,
- vorgesehene Filterliste für ausgehenden Verkehr und daraus resultierende propagierte Protokollverbindungen,

Copyright © Fraunhofer IESE 2003

```
Interface ethernet 5
-------------------------------------------------
Defined at:     102
Referenced at:  --
Description:    LAN SIGMA-Verwaltungsnetz
Enable state:   (implicitly) enabled
Address(es):    172.30.128.1
AccessGrp OUT:  105
Subnets reached:
                172.30.128.*
                  Node 172.30.128.45   ip
                  Node 172.30.128.2   ip

Accepted connections:
ip *.*.*.*  *.*.*.*

Propagated connections:
[0,2-255] 153.96.133-135.*  172.30.128.*
[0,2-255] 172.30.1.*  172.30.128.[2,45]
[0,2-255] 172.30.200.11  172.30.128.*
[0,2-255] 172.30.200.2  172.30.128.2
[0,2-255] 172.30.200.5  172.30.128.45
icmp *.*.*.*  172.30.128.*
```

Abbildung 18 Typische Informationen des Prüfmoduls 'Connectivity' zu Interface-Merkmalen

- Zeilennummer der Router-Konfigurationsbeschreibung, in der das Interface deklariert wird,
- Zeilennummern der Router-Konfigurationsbeschreibung, in denen auf das Interface verwiesen wird.

Abbildung 18 zeigt eine typische Ausgabe des Prüfmoduls bezüglich Interface-Merkmalen. Bemerkenswert sind vor allem die Abschnitte «Accepted Connections» und «Propagated Connections». Sie geben in hochverdichteter Form an, welche IP-Protokolle (aus dem Protokollnummern-Bereich 0…255) als eingehender bzw. ausgehender Verkehr vom Interface weitergeleitet werden. Um dies zu berechnen, müssen unterschiedlichste Konfigurationsklauseln ausgewertet werden, darunter Interface-Deklarationen, Access-Listen sowie statische Routen. Die beiden Zeilen

```
[0,2-255] 172.30.1.*  172.30.128.[2,45]
icmp *.*.*.*  172.30.128.*
```

(vgl. Abbildung 18) sind zum Beispiel wie folgt zu deuten:

Copyright © Fraunhofer IESE 2003

- Alle Nachrichten, deren Ursprungs-IP-Adresse mit '172.30.1' beginnt und deren Ziel-IP-Adresse entweder '172.30.128.2' oder '172.30.128.45' ist, können passieren, solange das Nachrichtenprotokoll eine Protokollnummer ungleich 1 (entweder 0 oder 2-255) hat. (Die Protokollnummer 1 bezeichnet das Protokoll ICMP.)

- Für das zuvor ausgenommene ICMP-Protokoll gilt: ICMP-Nachrichten sind von beliebigen Ursprüngen zu allen Zielen im Subnetz '172.30.128.*' zulässig.

Die übrigen Zeilen der beiden Positiv-Listen «Accepted Connections» und «Propagated Connections» sind in analoger Weise zu deuten.

Lines

Die Merkmale der Administrationszugänge des Routers — sogenannte Line Interfaces — werden in ähnlichem Format wie die übrigen Schnittstellen protokolliert. Abbildung 19 zeigt ein typisches Beispiel.

Subnets

Das Prüfmodul ermittelt die Adreßbereiche von Subnetzen, die in der Router-Konfigurationsbeschreibung ausdrücklich erwähnt werden, sei es in Zusammenhang mit Interface-Adressen, bei der Festlegung statischer Routen oder als Ursprünge bzw. Ziele von ACL-Filterregeln. Da ein Sicherheitsrevisor in der Regel die Adressen der wichtigsten umgebenden Subnetze kennt, ist die systematische

```
Line console 0
-------------------------------------------------
Defined at:     Line 198
Referenced at:  --
Enable state:   enabled
AccessGrp IN:   1

Accepted connections:
ip 151.200.19.52  *.*.*.*
ip 164.47.133.2  *.*.*.*
ip 217.31.34.129  *.*.*.*

Propagated connections:
ip *.*.*.*  *.*.*.*
```

Abbildung 19 Typische Informationen des Prüfmoduls 'Connectivity' zu Line-Merkmalen

Copyright © Fraunhofer IESE 2003

```
Subnet 175.16.*.*
-------------------------------------------------
Subnets:
                175.16.128.*
                175.16.1.*
Reached via:
                serial 0 (WAN-Verbindung zum Provider TRANSKOM)
```

Abbildung 20 Typische Subnetz-Informationen des Prüfmoduls 'Connectivity'

Aufstellung hilfreich, um die einzelnen Router-Schnittstellen gewissen funktionalen Bereichen zuzuordnen.

Die protokollierten Subnetz-Beschreibungen umfassen

- den Adreßbereich des Subnetzes,
- das nächstumfassendere Supernetz, das dieses Subnetz enthält (soweit vorhanden),
- die nächstkleineren Subnetze, die im Adreßbereich dieses Subnetzes enthalten sind (soweit vorhanden),
- den Interface-Namen, über den das Subnetz erreicht werden kann,
- die Interface-Klartextbeschreibung (soweit vorhanden).

Abbildung 20 zeigt ein Beispiel für eine typische Subnetz-Beschreibung.

Nodes

Soweit in der Router-Konfigurationsbeschreibung einzelne IP-Adressen ausdrücklich erwähnt werden, handelt es sich vermutlich um Netzknoten — in der Regel sind dies Rechner oder IP-Router. Das Prüfmodul erstellt eine Liste aller IP-Nodes und liefert dazu folgende Informationen:

- IP-Adresse des Knotens
- Subnetz, dem der Knoten angehört
- Interface, über das der Knoten erreichbar ist
- IP-Protokolle und Ports, über die der Knoten ansprechbar ist.

Copyright © Fraunhofer IESE 2003

```
Node 217.31.32.81
-------------------------------------------------
Member of:
                217.31.32.80-87
Reached via:
                ethernet 1
Ports:
                www
Protocols:
                ip tcp icmp
```

Abbildung 21 Typische Informationen des Prüfmoduls 'Connectivity' zu Netzknoten

Abbildung 21 zeigt eine typische Prüfmodulausgabe. Im vorliegenden Fall legt der WWW-Port den Schluß nahe, daß es sich bei dem Knoten um einen WWW-Server handelt.

Access Control Lists

Zugriffskontrollisten (Access [Control] Lists, ACLs) gehören zu den am schwierigsten zu analysierenden Konfigurationsmerkmalen eines Routers. Das Prüfmodul versucht, die Filterwirkung solcher Listen durch Transformation in ein Whitelist-Format (d.h. die erschöpfende Liste aller zulässigen Verbindungen) zu verdeutlichen. Daneben prüft 'Connectivity', ob alle ACLs auch korrekt deklariert und verwendet werden, welche Zusammenhänge zwischen einzelnen ACL-Klauseln bestehen und ob eine ACL wirkungslose Klauseln enthält. Zu den ermittelten Daten zählen folgende Punkte:

- Typ der ACL (standard oder extended)
- Korrekte Verwendung (Tippfehlerschutz!):
  - Jede definierte ACL wird auch verwendet
  - Jede verwendete ACL wird auch definiert
- Explizites Rücksetzen, d.h. Löschen, der ACL vor Deklaration?
- Wechselwirkungen der ACL-Regeln untereinander

Der Prüfbefund wird in Form von Annotationen in der Befunddatenbank abgelegt und ist in der Ausgabe direkt den entsprechenden Konfigurationsklauseln zugeordnet.

Copyright © Fraunhofer IESE 2003

Vertiefende Analysen zu ACLs oder Interfaces lassen sich mittels der Utility 'black-white' (siehe Kapitel 7.1) durchführen. Das Modul 'Connectivity' stellt dafür die relevanten Rohdaten im XML-Format bereit.

4.5 Modul 'SNMP'

Das SNMP-Modul überprüft die Konfigurationseinstellungen des Routers für das Simple Network Management Protocol (SNMP). SNMP dient dazu, relevante Zustandsinformationen in der Management Information Base (MIB) von Netzwerkkomponenten abzufragen und zu ändern oder außergewöhnliche Ereignisse (Traps) im Netzwerk an einen Management-Knoten zu signalisieren. In großen, ortsübergreifenden Netzwerken ist SNMP ein unverzichtbarer Quasi-standard.

SNMP bietet sicherheitskritische Funktionalität, denn es ermöglicht ein Ausspionieren, gegebenenfalls gar ein Umkonfigurieren des Routers von einem entfernten Management-Knoten aus. Daher bedürfen die SNMP-Einstellungen einer besonderen Sorgfalt.

Prüfvorgaben

Das Prüfmodul 'SNMP' ist nicht parametrisierbar. Alle Prüfkriterien sind derzeit durch den Modulcode fest vorgegeben.

Eine besondere Erschwernis ergibt sich bei den Prüfungen dadurch, daß IOS drei verschiedene SNMP-Versionen unterstützt, die sich hinsichtlich Authentisierungs- und Berechtigungskonzept voneinander unterscheiden. Das Prüfmodul kann SNMPv1, -v2 sowie v3-Klauseln auswerten, notfalls auch eine Kombination von Klauseln, die sich auf unterschiedliche Versionen beziehen.

Grundsätzlich steht es einem Anwender frei, eine gemischte SNMP-Konfiguration zu deklarieren, die unterschiedliche SNMP-Versionen gleichzeitig unterstützt. Dies führt unter Umständen zu schwer vorhersehbaren Effekten, deren ganze Tragweite sich nur durch Ausprobieren ermitteln läßt.[10] Aufgrund fehlender technischer Unterlagen war es nicht möglich, mit Hilfe des Prüfmoduls umfassende Vorhersagen bezüglich des Router-Verhaltens zu berechnen. Im Falle gemischter SNMP-Konfigurationen liefert das SNMP-Prüfmodul daher nur eine grobe Analyse zur Korrektheit der SNMP-Klauseln.

10 In eigenen Versuchen beobachteten wir bei gemischten Konfigurationen zum Teil ein recht merkwürdiges Verhalten des Routers. Unklar blieb, ob es sich dabei um eine Zufälligkeit des spezifischen Router-Typs oder eine verläßliche Eigenschaft der verwendeten IOS-Version handelt. Aufgrund solcher Erfahrungen raten wir von der Verwendung gemischter Konfigurationen ab.

Copyright © Fraunhofer IESE 2003

Authentisierung, Autorisierung und Funktionsumfang

SNMP verfügt — vor allem in der Version 3 — über verschiedene Mechanismen, um das Mißbrauchspotential zu verringern. Router-Konfigurationen sollten das ganze Spektrum der Authentisierungs- und Autorisierungsmöglichkeiten ausschöpfen, um den Zugriff auf den Router bestmöglich einzuschränken. Nicht benötigte Funktionalität sollte am besten vollständig deaktiviert werden.

In bezug auf Traps — spontane Mitteilungen des Routers über besondere Betriebszustände — ist es ratsam, sehr wichtige Mitteilungen möglichst nicht zu deaktiveren. Umgekehrt sollten häufige, weniger relevante Ereignisse nur sehr sparsam signalisiert und daher bevorzugt deaktiviert werden.

Das Prüfmodul analysiert daher vor allem folgende Punkte:

- Wurde der SNMP-Server deaktiviert?
- Sind die verwendeten Community Strings («SNMP Paßwörter») hinreichend schwierig zu erraten?
- Wird von sicheren Authentisierungs- und Verschlüsselungsmöglichkeiten bestmöglich Gebrauch gemacht?
- Werden die Sichten auf SNMP-Informationen durch Views eingeschränkt?
- Wird der Zugriff möglichst auf READONLY beschränkt?
- Sind SNMP-Zugriffe durch Access-Listen eingeschränkt?
- Ist ein System Shutdown via SNMP ausgeschlossen?
- Welche SNMP-Traps sind aktiviert oder deaktiviert?

Manager-Einstellungen

Router zählen in der Regel zu den verwalteten (managed), nicht zu den verwaltenden (managing) Objekten. Es ist daher unüblich, einen Router als SNMP-Manager zu konfigurieren. Das Prüfmodul warnt, wenn der Router als Manager konfiguriert ist, somit also andere Netzwerkkomponenten womöglich umkonfigurieren kann.

Copyright © Fraunhofer IESE 2003

Verschiedenes

Insgesamt sind die verfügbaren IOS-Klauseln zur Konfiguration der SNMP-Einstellungen recht verwickelt und voller Seiteneffekte, deren Wirkung nur sehr vage dokumentiert ist. Um Probleme zu vermeiden, sollte man sich an die von Cisco empfohlenen Vorgehensweisen halten. Das Werkzeug prüft unter anderem:

- Sind SNMP-Klauseln in der von Cisco empfohlenen Reihenfolge deklariert?
- Gibt es veraltete oder (wegen Überdeckung) unwirksame SNMP-Klauseln?
- Werden ungebräuchliche Nachrichtenlängen (packet sizes) verwendet?

Bemerkungen

Aus technischer Sicht wäre es ohne weiteres vorstellbar, die Prüftiefe des SNMP-Moduls noch erheblich zu verbessern. Derzeit scheitert dies vor allem an den unzureichenden Informationen, die Cisco zum Thema «SNMP-Konfiguration unter IOS» zur Verfügung stellt. Wünschenswert wäre insbesondere, das genaue Zusammenspiel zwischen SNMPv1, -v2 und -v3 im Prüfmodul genau nachzubilden, um exaktere Vorhersagen über gemischte Konfigurationen treffen zu können.

4.6 Modul 'NTP'

Das NTP-Prüfmodul überprüft die Konfigurationseinstellungen des Routers für das Network Time Protocol (NTP). NTP dient dazu, die lokalen Systemuhren von Netzwerkkomponenten miteinander abzugleichen. Synchron laufende Uhren sind wichtig, denn sie erleichtern die Überwachung und Fehlerbehebung in Kommunikationsnetzen erheblich.

Das Prüfmodul 'NTP' baut auf Analysen des Prüfmoduls 'Connectivity' auf (vgl. Kapitel 4.4) und kann daher nur im Verbund mit diesem aktiviert werden. Ist 'Connectivity' für einen Prüflauf nicht aktiviert, so wird es für das abhängige Modul 'NTP' automatisch nachgeladen.

Prüfvorgaben

Die Prüfschritte des NTP-Moduls sind nicht parametrisierbar, sondern im Quellcode des Moduls festgelegt. Eine Einflußnahme auf die Prüfkriterien ist also nur durch das Umprogrammieren des Moduls möglich.

Copyright © Fraunhofer IESE 2003

Das Prüfmodul identifiziert zum einen mögliche Schwächen und Fehler in der NTP-Konfiguration. Es weist auf widersprüchliche NTP Konfigurationsklauseln und auf das Fehlen gewisser Standardvorgaben hin.

Zum anderen erstellt es eine Liste, anhand derer man leicht erkennen kann, von welchem Host der Router seine Systemzeit bezieht und an welche er diese weitergeben kann. Dabei wird auch die Möglichkeit berücksichtigt, NTP-Nachrichten mittels Broadcast zu verschicken.

Widersprüche und logische Fehler

Während der Überprüfung der einzelnen Zeilen der Konfigurationsbeschreibung beurteilt das Modul die Konsistenz der Anweisungen. Es wird überprüft, ob alle Schlüssel und Access-Listen, auf die zugegriffen wird, auch definiert sind. Umgekehrt wird geprüft, ob auf alle deklarierten Schlüssel und Access-Listen auch bezug genommen wird. Das Redefinieren oder Löschen eines zuvor deklarierten Schlüssels ist bei einem vorgefertigten Skript nicht notwendig; es deutet auf eine unbedachte lokale Änderung der Konfigurationsbeschreibung hin — ein potentieller Konfigurationsfehler.

Zudem wird während dieser Phase auch auf fehlende Konfigurationsparameter hingewiesen. Beinhalten Schlüssel und Access-Listen-Kommandos Default-Werte, wird dies angemerkt und in der späteren Analyse berücksichtigt. Sind Kommados in einem IOS-Modus nicht zulässig, wird eine Warnung ausgegeben.

Viele Fehler ergeben sich erst aus dem logischen Zusammenwirken aller Konfigurationsklauseln. Wurde zum Beispiel die Authentisierung durch Schlüssel konfiguriert, so ist das Kommando 'ntp-server' ohne die Angabe eines Schlüssels wirkungslos. Das Modul versucht, solche Unstimmigkeiten auszuweisen.

Liste der IP-Adressen und Interfaces

Nach dem Prüflauf werden die entsprechenden IP-Adressen aller Server und Peers aufgelistet, mit denen der Router eine NTP-Kommunikation führt. Zudem werden die Interfaces angezeigt, über die Broadcast-Nachrichten für das Zeitprotokoll ein- bzw. ausgehen. Die Daten, die hier aufgelistet werden, sind bereits auf Plausibilität überprüft. Zudem werden die mittels 'ntp access-group' angewandten Access-Listen berücksichtigt. Allerdings muß der Netzwerkadministrator die Listen der beteiligten Interfaces überprüfen: Hier muß der Port 123 TCP oder UDP an mindestens einem Interface freigeschaltet sein, damit die NTP-Anfragen nicht durch diese Restriktion verworfen werden. Eine automatische Kontrolle kann hier nicht vorgenommen werden, da die Netzumgebung dem Werkzeug nicht bekannt ist und deshalb auch nicht entscheidbar ist, *welches* Interface für NTP freigeschaltet werden muß.

Copyright © Fraunhofer IESE 2003

Auch das direkte Anschließen einer Uhr an den Router wird berücksichtigt. Zudem wird, falls die Authentisierungsmethode mittels Schlüsseln gewählt wurde, eine Liste der möglichen und der vertrauenswürdigen Schlüssel angezeigt. Letztere müssen, falls diese Authentisierungsmethode aktiviert wurde, einem NTP-Server mitgeteilt werden, damit der Router die ankommenden Nachrichten auch akzeptiert und seine eigene Uhr entsprechend umstellt. Somit ist vom Netzwerkadministrator nachträglich die Aufzählung aller Peers bzw. aller Server zu nehmen und auf den entsprechenden Maschinen die Schlüsselkonfiguration zu überprüfen. Hierbei sind die Listen der lokal deklarierten und genutzten Authentisierungsschlüssel sehr hilfreich.

Der Router als NTP-Server

Ein Cisco Router antwortet grundsätzlich auf NTP-Anfragen anderer Hosts mit seiner Software-Uhrzeit. Wurde der Router allerdings vorher nicht durch eine andere Zeitquelle synchronisiert, so besitzt die NTP-Nachricht das Stratum 0. In einem IP-Paket gilt dies als Zeichen, daß die Zeit unsynchronisiert ist. Ein Client wird diese Mitteilung ignorieren. Ist der Router allerdings durch NTP synchronisiert worden oder wurde er mittels des Befehles 'ntp-master' dazu erklärt, so fungiert er auf allen Interfaces als NTP-Server, sofern die Schnittstelle nicht explizit davon ausgenommen wurde. Als versendete Zeit dient hier die Software-Uhr (Clock) des Routers. Das Prüfmodul erstellt für jede Schnittstelle eine Übersicht der möglichen Hosts, die eine Zeitanfrage stellen können. Dies ist durch die Access-Listen der Interfaces bzw. die globale NTP-ACL festgelegt.

Unter dem Bookmark "NTP accepted" ist für jedes Interface getrennt aufgelistet, welchen Hosts es möglich ist, unter Einbeziehung der aktuellen Konfiguration den Router als NTP-Server zu nutzen. Natürlich werden auch hier alle Router-Access-Listen in Betracht gezogen, allerdings muß der Systemadministrator nachträglich überprüfen, ob bei den Interfaces der beteiligten Netzkomponenten Port 123 TCP oder UDP freigeschaltet ist, da die NTP-Antwortnachrichten den Router sonst nicht verlassen können. Hierfür ist allerdings die Kenntnis der Netzwerkumgebung erforderlich, die Prüfung kann daher nicht vollautomatisch erfolgen.

Für eine erfolgreiche Nutzung des Routers als NTP-Server ist im übrigen ein Abgleich der Authentisierungsschlüssel notwendig, da der Server den Schlüssel kennen muß, den der Client verlangt. Auch dies erfordert Kontextinformationen, die aus der lokalen Konfigurationsbeschreibung nicht ableitbar sind.

Logische Kontexte zur Erhöhung der Übersicht

Das NTP-Modul legt zusätzlich einen separaten Kontext an, dem alle Anweisungen zugeordnet werden, die sich auf den Zeitdienst beziehen. Die Kontextan-

Copyright © Fraunhofer IESE 2003

sicht zeigt ausschließlich diese Zeilen, unabhängig davon, wie sie über die Konfigurationsbeschreibung verstreut sind. So kann der versierte Nutzer schnell die Auswirkungen aller NTP-Befehle erkennen. Ein weiterer logischer Kontext faßt alle Befehle zusammen, die sich auf Authentisierungsschlüssel beziehen (falls vorhanden).

Anmerkung zu NTP

NTP beeinflußt normalerweise nur die Software-Uhr des Routers (Clock). Hat dieser noch eine Hardware-Uhr (Calendar), so kann auch sie von NTP synchronisiert werden (durch den Befehl 'ntp update-calendar'). Da beim Hochfahren automatisch die Hardware-Zeit vom Calendar in die Software-Uhr kopiert wird hat dies den Vorteil, daß der Router auch nach einem Reboot wieder die korrekte Zeit besitzt.

Wird der Calendar nicht von NTP aktualisiert, so kann nach einer gewissen Laufzeit des Gerätes eine beträchtliche Gangabweichung zwischen Clock und Calendar entstehen. Dies ist kein Problem, solange der Router in Betrieb ist. Muß er aber neu hochgefahren werden, übernimmt die Software-Uhr automatisch die womöglich ziemlich falsche Calendar-Zeit. Da ein Host bei der Verwendung von NTP nur dann seine Uhr umstellt, wenn die lokale Zeit nicht zu sehr von der NTP-Zeit abweicht, kann dies dazu führen, daß der Router nach dem Neustart nicht mehr synchronisiert wird und so unbemerkt außerhalb des NTP-Verbunds steht.

Aus diesem Grund ist es ratsam, die Uhr eines jeden Routers nach einem Reboot oder bei NTP-Aktivierung mittels 'show clock' zu überprüfen und bei falscher Zeit manuell mit 'clock set' auf die aktuelle Zeit zu setzen.

4.7 Modul 'AAA'

Die AAA-Funktionen unter IOS regeln Authentisierung, Autorisierung und Accounting. Das Prüfmodul 'AAA' analysiert die Widerspruchsfreiheit, Vollständigkeit und Sicherheit der AAA-Konfigurationsvorgaben.

Prüfvorgaben

Sämtliche AAA-Prüfkriterien sind im Programmcode des Prüfmoduls festgelegt; parametrisierbare Prüfvorgaben sind nicht vorgesehen.

Das Modul analysiert derzeit nur Authentisierungs- und Autorisierungsmerkmale. Der dritte Bereich — Accounting — bleibt unberücksichtigt, da er nicht als sicherheitskritisch einzustufen ist. Die wichtigsten Prüfungkriterien sind:

Copyright © Fraunhofer IESE 2003

- Werden AAA-Klauseln in der falschen Reihenfolge spezifiziert?
- Ist die Konfiguration der verwendeten AAA-Funktionen unvollständig oder enthält sie Widersprüche?
- Werden undeklarierte AAA-Merkmale genutzt, oder werden unbenutzte AAA-Merkmale deklariert?
- Werden unsichere AAA-Merkmale konfiguriert?
- Werden IOS-Funktionen konfiguriert, die nicht mit AAA kompatibel sind?
- Werden veraltete Authentisierungs- oder Autorisierungsmechanismen verwendet?

Authentisierung

Mittels AAA können verschiedene Aktivitäten authentisiert werden, darunter Login, Dial-in ("PPP") oder auch der Wechsel in den sogenannten privilegierten Modus ("enable"). Per Konfiguration kann jeder dieser Aktivitäten eine Liste von Authentisierungsmethoden zugeordnet werden, die der Reihe nach probiert werden, um die Identität des Nutzers zu prüfen. Die Methoden können wahlweise auf Line-Paßwörter, die lokale User-Datenbank des Routers oder auf externe Authentisierungsserver vom Typ RADIUS, TACACS+ oder Kerberos zurückgreifen. Als Fallback kann sogar eine Null-Authentisierung eingetragen werden.

Das Modul prüft, ob die deklarierten Methodenlisten tatsächlich auch verwendet werden und ob sie sicherheitstechnisch unbedenklich sind. Soweit auf Line-Paßwörter oder auf die User-Datenbank zurückgegriffen wird, prüft das Modul, ob entsprechende Paßwörter bzw. paßwortgeschützte lokale Kennungen auch wirklich konfiguriert sind. Soll ein externer Server verwendet werden, so analysiert das Prüfmodul, ob ein Server des entsprechenden Typs konfiguriert ist.

Sofern bestimmten Aktivitäten eine Authentisierungsmethodenliste zugeordnet ist, muß diese auch definiert sein. Das Prüfmodul stellt sicher, daß nicht versehentlich eine leere oder undefinierte Methodenliste verwendet wird.

Autorisierung

AAA ermöglicht es, bestimmte IOS-Befehlsgruppen einer gesonderten Autorisierung zu unterstellen. Autorisierung kann zum Beispiel für Dial-in, Shell-Zugriff, Reverse Telnet oder auch für ein sogenanntes Privilege Level festgelegt werden. Ähnlich wie bei der Authentisierung werden Autorisierungsmethodenlisten

Copyright © Fraunhofer IESE 2003

definiert, mit deren Hilfe die Berechtigung zur Nutzung einzelner Befehlsgruppen vom Router überprüft wird. Sowohl lokale Autorisierung anhand der User-Datenbank wie Server-gestützte Autorisierung mittels RADIUS, TACACS+ oder Kerberos sind konfigurierbar.

Das Prüfwerkzeug stellt sicher, daß alle Betriebsmittel, die zur Anwendung einer Autorisierungsmethode erforderlich sind, lokal auch konfiguriert wurden. Es prüft auch, ob die verwendeten Methodenlisten definiert sind und ob alle definierten Methodenlisten auch wirklich verwendet werden. Auch die innere Widerspruchsfreiheit der Methodenlisten wird überprüft.

Externe Authentisierungs- und Autorisierungsserver

Sofern sich Authentisierung oder Autorisierung auf einen externen Server stützen, muß dieser entsprechend konfiguriert sein. Das Prüfmodul analysiert, inwieweit RADIUS, TACACS+ oder Kerberos benötigt werden. Wenn Bedarf für einen solchen Server besteht, wird überprüft, ob eine sichere Kommunikation mittels RADIUS-, TACACS- bzw. Kerberos-Protokoll gewährleistet ist. Zu den maßgeblichen Attributen, die zu konfigurieren sind, zählen Schlüssel zur Authentisierung des Servers sowie einheitliche Ursprungsadressen.

Da die Kommunikation mit einem Authentisierungs- oder Autorisierungsserver leicht einmal gestört sein kann, empfiehlt es sich, mehrere unabhängige Server des gleichen Typs vorzusehen, die wahlweise kontaktiert werden können. Dazu können sogenannte Server-Gruppen konfiguriert werden. Das Prüfmodul analysiert, ob alle verwendeten Gruppen-Namen auch definiert sind und ob alle Server einer Gruppe zuvor auch konfiguriert wurden.

Bei Verwendung eines TACACS-Servers setzt AAA das TACACS+-Protokoll voraus. Ältere Protokollvarianten — TACACS bzw. Extended TACACS — sind nicht mit AAA verträglich. Das Prüfmodul warnt, wenn AAA in Kombination mit einem der veralteten Protokolltypen konfiguriert worden ist.

Bemerkungen

Das Modul 'AAA' ist derzeit eines der aufwendigsten Prüfmodule. Dies hängt zum einen damit zusammen, daß Authentisierung und Autorisierung ein komplexes Themengebiet ist.[11]

Zum anderen hat es den Anschein, daß die AAA-Mechanismen noch relativ unausgereift sind und daher einem ständigen Wandel unterliegen. Cisco hat

11 Dies zeigt sich schon daran, daß etwa 30 Pattern erforderlich waren, um die für AAA relevanten Konfigurationsklauseln zu erfassen. Zur Analyse müssen neben AAA-Befehlen auch Interface-, Line-, User- und Server-Deklarationen analysiert werden.

Copyright © Fraunhofer IESE 2003

sowohl Syntax als auch Funktionsweise innerhalb weniger IOS-Generationen mehrfach geändert. Dabei haben sich offensichtliche Fehler und Widersprüche in die AAA-Dokumentation eingeschlichen, und auch die innere Logik der AAA-Direktiven hat unter den zahlreichen Modifikationen erheblich gelitten.

Die Prüfung der AAA-Funktionalität ist zwangsläufig unvollständig, sobald externe Server beteiligt sind. In diesem Fall hängt die Sicherheit nämlich ebenso sehr von der Konfiguration des Servers wie der des Routers ab. Neben der Analyse der Router-Konfiguration ist es daher unverzichtbar, auch die verwendeten Server in die Betrachtungen mit einzubeziehen. Dies kann das Prüfwerkzeug nicht leisten.

4.8 Modul 'Logging'

Cisco IOS generiert Alarm-, Informations- und Debug-Meldungen zu relevanten Hardware- und Software-Ereignissen des Routers. Diese Meldungen können zur Konsole, in lokale Puffer oder zu einem externen Server gelenkt werden. Dies liefert ein Protokoll («Log») des Routerbetriebs — ein wichtiger Baustein für ein umfassendes IT-Sicherheitskonzept.

Das Prüfmodul 'Logging' prüft die für Logging relevanten Konfigurationseinstellungen auf Korrektheit, Widerspruchsfreiheit und Vollständigkeit. Es erstellt eine nach Bereichen geordnete Übersichtsdarstellung aller relevanten Parameter.

Prüfvorgaben

Sämtliche Prüfkriterien zum Bereich 'Logging' sind im Programmcode des Prüfmoduls festgelegt; parametrisierbare Prüfvorgaben sind nicht vorgesehen.

Insgesamt bietet IOS fünf verschiedene Möglichkeiten zur Logdatenerfassung: drei lokale und zwei netzbasierte Varianten. Die Protokollierung erfolgt wahlweise

- auf die Router-Konsole,
- auf eine Terminal-Line–Schnittstelle (Monitor),
- in einen lokalen Pufferspeicher,
- nach einem entfernen SYSLOG-Server oder
- nach einem entfernten SNMP-Server.

Copyright © Fraunhofer IESE 2003

Für eine systematische Überwachung des Routerbetriebs sind vor allem die drei letztgenannten Varianten empfehlenswert[12]; sie werden besonders kritisch analysiert. Alle fünf Logging-Möglichkeiten können parallel und unabhängig von einander genutzt werden. Für jede der Varianten werden folgende Punkte überprüft:

- Ist das Logging-Verfahren aktiviert und sind alle Konfigurationsvoraussetzungen für eine ordnungsgemäße Protokollierung erfüllt?
- Ab welcher Dringlichkeitsstufe werden Meldungen protokolliert (IOS unterscheidet die folgenden Dringlichkeitsstufen nach steigender Priorität: debugging, informational, notifications, warnings, errors, critical, alerts, emergencies)?
- Ist die maximal zulässige Meldungsrate beschränkt?
- Im Falle von SYSLOG: Unter welchem Facility-Namen werden die Meldungen an den Server übermittelt?

Daneben werden verschiedene allgemeine Prüfpunkte analysiert:

- Sind die Meldungen mit Zeitstempeln versehen?
- Wie sind die Meldungspuffer dimensioniert?
- Sind zusätzliche Logging Services aktiviert (z. B. Line Card Logging, VIP Card Logging, BGP oder EIGRP Logging, MPLS Logging)?
- Werden Zugriffskontrollisten (ACLs) mittels 'log'-Attribut überwacht, und wird die Dringlichkeitsstufe entsprechender Meldungen bei der Protokollierung überhaupt berücksichtigt?
- Sind alle Konfigurationsklauseln syntaktisch korrekt, und liegen deren Parameterwerte innerhalb des zulässigen Wertebereichs?
- Gibt es sich widersprechende Konfigurationsklauseln?

12 Eine direkte Protokollierung auf ein Ausgabegerät ist aus verschiedenen Gründen nur von sekundärem Interesse: Erstens ist eine nachträgliche Aufbereitung mittels Textverarbeitungswerkzeugen nur noch bedingt möglich. Zweitens kann der Darstellungsbereich einer Konsole überlaufen, was zu Datenverlusten führt. Und drittens vermischen sich die Logmeldungen mit Eingaben des Benutzers und sonstigen Ausgaben des IOS. Dies alles erschwert die Analyse.

Copyright © Fraunhofer IESE 2003

SNMP Logging

Für die Übermittlung von Logdaten an einen SNMP-Server dienen sogenannte SNMP Traps. Damit Trap-Meldungen korrekt übermittelt und empfangen werden können, sind eine Reihe von Konfigurationseinstellungen vorzunehmen, sowohl Logging betreffend, als auch SNMP.

Das Prüfmodul 'Logging' beschränkt sich im wesentlichen auf die Überprüfung der Einstellungen bezüglich Logging. Die SNMP-seitigen Konfigurationseinstellungen bleiben weitgehend unberücksichtigt. Der Anwender kann diese mit dem Prüfmodul 'SNMP' (vgl. Kapitel 4.5) analysieren. Um die Prüflogik nicht zu duplizieren, verzichtet das Modul 'Logging' auf eigenen SNMP-Prüfregeln.

Befundaufbereitung

Die Prüfpunkte werden zu logischen Kontexten gruppiert. Das Prüfergebnis wird unter dem Bookmark 'Logging' zusammengefaßt.

Befunde zum Konsolen- und Terminal-Line-Logging werden, soweit es sich nicht um eindeutige Konfigurationsfehler handelt, eher wertneutral gemeldet. Ist die entsprechende Protokollierungsvariante abgeschaltet, so wird dies nicht als besorgniserregend eingestuft. Demgegenüber setzt das Prüfmodul voraus, daß der lokaler Logdatenpuffer sowie eine externe Logdatenerfassung (SYSLOG oder SNMP) aus Sicherheitsgründen aktiviert sein sollten. Fehlen entsprechende Konfigurationsklauseln, so wird dieser Befund wenigstens unter der Kategorie 'CHECK' ausgewiesen.

Konfigurationsklauseln zur Dimensionierung von Puffergrößen und Übertragungsraten werden in der Regel der Kategorie 'CHECK' zugeordnet, unabhängig von den verwendeten Attributwerten. Es ist Aufgabe des Anwenders, zu beurteilen, ob angemessene Größen verwendet wurden — sinnvolle Einstellung sind in hohem Maße anwendungsabhängig und entziehen sich daher einer automatischen Bewertung.

Bemerkungen

Das Prüfmodul verzichtet bewußt auf konfigurierbare Sollwert-Vorgaben. Die Prüfung beschränkt sich vielmehr darauf, die innere Konsistenz der Logging-Konfiguration zu analysieren und die vorhandenen Einstellungen in übersichtlicher Gruppierung darzustellen. Dies sollte eine ausreichende Basis für eine Beurteilung der Korrektheit und Sinnhaftigkeit der Konfiguration liefern. Da sinnvolle Sollwerte sehr kontextspezifisch auf den konkreten Anwendungsfall zugeschnitten werden müssen, lohnt deren Spezifikation kaum — eine Vorgabe birgt eher die Gefahr von Spezifikationsfehlern.

Copyright © Fraunhofer IESE 2003

4.9 Modul 'Passwords'

Auf einem IOS-Router können verschiedene Paßwörter — in einigen Befehlen auch «secrets» genannt — konfiguriert werden, zum Beispiel Login-Paßwörter, Enable-Paßwörter oder auch FTP-Paßwörter. Um die Paßwortvorgaben gegen Ausspähung zu schützen, können Paßwörter in der Konfiguration wahlweise auch in verschlüsselter Form hinterlegt werden: IOS bietet dafür zwei Verschlüsselungsvarianten, Nr. 5 und Nr. 7. Die 5er-Verschlüsselung ist kryptographisch sicher, das 7er-Verfahren dient hingegen nur der Verschleierung, denn eine Entschlüsselung ist problemlos möglich.

Das Prüfmodul 'Passwords' untersucht die Routerkonfiguration auf unsichere Paßwörter oder Paßwörter, die unnötig schwach verschlüsselt wurden, sowie auf fehlende Aktivierung des Paßwort-Verschlüsselungsdienstes.

Prüfvorgaben

Die Prüfkriterien zum Bereich 'Passwords' sind überwiegend im Programmcode des Prüfmoduls festgelegt; lediglich die Kandidatenliste für den Test «Suche nach trivialen Passwörtern» ist mittels 'Configuration/Passwords.conf' frei konfigurierbar. Die wichtigsten Prüfpunkte des Prüfmoduls sind:

- Sind alle in 'password'- bzw. 'secret'-Klauseln enthaltenen Paßwortcodes zulässige Zeichenketten?
- Werden unsichere, d.h. leicht erratbare Paßwörter verwendet? Werden Mindeststandards hinsichtlich Paßwortlänge und -zeichenzusammensetzung eingehalten?
- Werden unverschlüsselte oder nur leicht verschlüsselte (Nr. 7) Paßwörter konfiguriert, wo statt dessen auch starke Verschlüsselung möglich wäre?
- Werden unverschlüsselte oder nur leicht verschlüsselte (Nr. 7) Paßwörter gleichzeitig auch als stark verschlüsselte (Nr. 5) Paßwörter verwendet, was die starke Verschlüsselung kompromittieren würde?
- Werden Paßwörter oder Secrets parallel zu anderen AAA-Authentisierungsmechanismen verwendet? (vgl. Kapitel 4.7)
- Ist der Paßwort-Verschlüsselungsdienst («service password-encryption») des Routers aktiviert, um Paßwortangaben im Konfigurationstext und in IOS 'show'-Kommandos zu verschleiern?
- Für welche der 'enable'-Ebenen 0 bis 15 sind Paßwörter konfiguriert?

Copyright © Fraunhofer IESE 2003

Alle Befunde werden unter dem Bookmark 'Authentication & Authorization', Unterpunkt 'Passwords and Secrets', zusammengefaßt.

Paßwortentschlüsselung

Paßwörter, die mit dem Verfahren Nr. 7 verschlüsselt wurden, lassen sich mit einem sehr einfachen Verfahren entschlüsseln. Das Prüfmodul nutzt dies für Wiederverwendungsprüfungen. CROCODILE gibt normalerweise das Entschlüsselungsergebnis zu jedem 7er-Paßwort als 'INFO'-Meldung aus, um so auf die Problematik solcher Paßwörter hinzuweisen.

Um bei seinen Analysen die Verschleierung von Paßwörtern aufrecht zu erhalten, kann der Anwender auf Wunsch die Klartextausgabe entschlüsselter Paßwörter unterdrücken. Dazu ist in der Datei 'Configuration/Passwords.conf' die Einstellung

```
DISCLOSE_PASSWORDS = NO
```

zu wählen. Die entsprechenden Meldungen werden daraufhin nicht mehr in den Analysebefund eingetragen. Der Anwender sollte sich jedoch darüber im klaren sein, daß dies nur geringen Schutz gegen flüchtige Ausspähung bietet: Das von CROCODILE verwendete Entschlüsselungsverfahren ist öffentlich bekannt; ein Angreifer kann es sich leicht im Internet beschaffen.

Die Default-Einstellung ist «DISCLOSE_PASSWORDS = YES», d.h. alle Entschlüsselungen werden im Prüfprotokoll offengelegt.

Triviale Nr. 5 Paßwörter

Das Verschlüsselungsverfahren Nr. 5 ist nach heutigem Kenntnisstand sicher; eine systematische Entschlüsselung erscheint unmöglich. Deshalb kann das Prüfmodul 'Passwords' die Güte der so verschlüsselten Paßwörter nicht ohne weiteres überprüfen. Die Tests beschränken sich daher darauf, einige offensichtlich mangelhafte Paßwortkandidaten auszuschließen.

Zunächst wird geprüft, ob eines der Klartext- oder 7er-Paßwörter in 5er-Verschlüsselung in der Konfiguration erscheint. In diesem Falle wäre die 5er-Verschlüsselung sinnlos, denn der Klartext dazu ist sofort gefunden.

Als nächstes wird getestet, ob das Chiffrat eines 5er-Paßworts der Verschlüsselung eines trivialen Klartexts entspricht. Die Liste der als trivial angesehenen Klartexte ist in der Konfigurationsdatei 'Configuration/Passwords.conf' in dem Attribut 'FORBIDDEN_PASSWORDS' als Liste hinterlegt; der Anwender kann diese Liste nach Gutdünken ergänzen oder verändern.

Copyright © Fraunhofer IESE 2003

Achtung: *Die Verschlüsselung Nr. 5 ist ein (absichtlich) sehr aufwendiges Verfahren. Das Durchprobieren von Klartextkandidaten — man bezeichnet dies auch als «Wörterbuchangriff» (Dictionary Attack) — erfordert daher relativ viel Rechenzeit. Daher empfiehlt es sich, die Liste 'FORBIDDEN_PASSWORDS' nicht allzu sehr auszudehnen.*

Zusätzlich zu den in 'FORBIDDEN_PASSWORDS' genannten Paßwortkandidaten wird noch der Hostname des Routers, falls er definiert ist, als mögliches Paßwort ausprobiert.

Der Anwender kann den Wörterbuch-Angriff auf 5er-Paßwörter unterdrücken, indem er in der Konfigurationsdatei 'Configuration/Passwords.conf' die Zeile

```
NO_5_CHECK = NO
```

einträgt. Dies vermindert die Laufzeit des Prüfmoduls. Wahlweise kann der Anwender aber auch einfach unnötigen Einträge aus der Kandidatenliste 'FORBIDDEN_PASSWORDS' entfernen.

Package 'Digest::MD5'

Das Nr. 5 Verschlüsselungsverfahren basiert auf MD5, einem kryptographischen Hash-Algorithmus der Firma RSA Data Security, Inc. Zur Implementierung der Verschlüsselung stützt sich das Prüfmodul 'Passwords' der Einfachheit halber auf ein fertiges Modul von CPAN, dem Comprehensive PERL Archive Network: Digest::MD5. Dieses Package stellt die benötigte Funktionalität auf bequeme Weise zur Verfügung. In der Regel sollte das Package in einer modernen PERL Standarddistribution enthalten sein.

Sollte in der lokalen PERL-Installation des Anwenders dieses Modul wider Erwarten fehlen, so erscheint bei Programmstart folgende Warnmeldung auf der Konsole:

```
*** Auxiliary::BSDcrypt -- Unable to locate CPAN Module 'Digest::MD5'
*** You may either try to install this module (see www.cpan.org)
*** or use your Perl unchanged: Just ignore this error message!
*** In the latter case, CROCODILE can't perform any No. 5 checks
*** Apart from that, CROCODILE's functionality remains unchanged.
```

Der Benutzer kann in diesem Fall die Programmausführung einfach fortsetzen; CROCODILE wird sich im wesentlichen unverändert verhalten. Lediglich solche Prüfpunkte des Moduls 'Passwords', die sich auf Nr. 5 verschlüsselte Passwörter beziehen, liefern keinen Befund. Alle übrigen Module behalten ihren vollen Funktionsumfang.

Copyright © Fraunhofer IESE 2003

Wahlweise kann der Anwender aber auch versuchen, das fehlende Modul 'Digest::MD5' von einem CPAN-Server kostenlos nachzuinstallieren. Eine mögliche URL dafür ist 'www.cpan.org'. Er sichert dadurch den vollen Prüfumfang des Moduls 'Passwords'.

Copyright © Fraunhofer IESE 2003

5 Emulation des Router Audit Tools (RAT)

Das Router Audit Tool (RAT) ist ein Werkzeug zur Überprüfung von IOS-Konfigurationen. Es wird vom Center for Internet Security (CIS) [CIS] im Internet zum kostenlosen Download angeboten [RAT2.0]. Ähnlich wie das CROCODILE-Modul 'CompoundPatterns' liest RAT frei konfigurierbare Prüfregel-Spezifikationen ein und wendet sie auf die zu prüfende Routerkonfiguration an. Das Prüfergebnis besteht aus einer Liste aller angewendeten Prüfschritte und deren jeweiligem Ergebnis 'PASS' oder 'FAIL'. Aus der Gewichtung der Prüfregeln und der Liste der Prüfergebnisse errechnet RAT eine Benchmark-Größe: Diese Kennzahl ist das Ergebnis des CIS Router Benchmarks (siehe dazu auch Abbildung 23 auf Seite 92).

RAT ist recht bekannt, und es wird von bedeutenden Organisationen gefördert, so zum Beispiel von der National Security Agency [NSA] oder dem SANS Institute [SANS]. Aufgrund dieser Förderung gibt es für RAT umfangreiche, gut ausgearbeitete Prüfregelsätze, die auch aktualisiert werden.

CROCODILE bietet ein spezielles Emulationsmodul, um sich diese Ressourcen zu erschließen. Das Modul emuliert RAT Version 2.0 in der CIS-Distribution vom März 2003. Mittels der RAT-Prüfregelsätze des CIS kann es den CIS-Benchmark nachbilden.

5.1 Emulationsmodul 'RATemulation'

Das Prüfmodul 'RATemulation' liest RAT-Prüfregel-Spezifikationen ein und wendet diese Prüfregeln auf den Untersuchungsgegenstand an. Damit ermöglicht 'RATemulation' frei konfigurierbare Prüfvorgaben in den Grenzen des RAT-Formats.

In seiner flexiblem Konfigurierbarkeit gleicht das Emulationsmodul dem Modul 'CompoundPatterns' (vgl. Kapitel 4.1), und tatsächlich entspricht die Mächtigkeit der spezifizierbaren Prüfregeln ziemlich genau dem Ausdrucksvermögen der sogenannten «elementaren Patterns» jenes Moduls (Compound Patterns sind jedoch noch mächtiger, vgl. Tabelle 2 auf Seite 51). Im Kern besteht jede RAT-Regel aus einem Pattern, das eine IOS-Klausel beschreibt. Ein Pattern ist entweder als 'required' oder als 'forbidden' gekennzeichnet, und ein Prüfschritt scheitert, wenn ein als 'required' deklariertes Pattern fehlt oder ein als

Copyright © Fraunhofer IESE 2003

'forbidden' gekennzeichnetes Pattern in der Routerkonfiguration nachgewiesen wird. Somit handelt es sich wie bei elementaren Patterns um Prüfregeln zu einzelnen Konfigurationszeilen: Prüfregeln für die Bewertung des Zusammenspiels mehrerer, unzusammenhängender IOS-Klauseln sind im RAT-Regelformat nicht darstellbar.[13]

Ähnlich wie bei 'CompoundPatterns' können RAT-Prüfregeln auf spezifische Konfigurationskontexte eingeschränkt werden (vgl. dazu Kapitel 4.1, Abschnitt «Kontexte» auf Seite 50), zum Beispiel auf den Interface- oder Line-Konfigurationsmodus. Diesbezüglich bietet RAT gegenüber 'CompoundPatterns' nichts neues.

Im Unterschied zu 'CompoundPatterns' sind die Prüfregeln für 'RATemulation' jedoch mit einer Reihe zusätzlicher Attribute versehen, darunter zum Bespiel 'Importance' (Schwere), 'Description' (Gegenstand des Prüfschritts), 'Reason' (Sicherheitsrelevanz), 'Fix' (Korrekturvorschlag bei Verletzung der Regel) sowie 'Warning' (Hinweise zu Einschränkungen der Regelanwendung). Solche Zusatzinformationen lassen sich nutzen, um den vom Werkzeug erzeugten Prüfbericht um Erläuterungen und Tips zu ergänzen. Dies macht den RAT-Fundus auch für CROCODILE interessant. Daher wurde unser Framework um ein RAT-Modul erweitert.

Die Fähigkeit, RAT-Regelsätze originalgetreu zu verarbeiten, erleichtert ehemaligen RAT-Anwendern den Umstieg auf CROCODILE. Sofern Anwender eigene, den lokalen Gegebenheiten angepaßte Prüfschritte für RAT spezifiziert hatten, können sie diese privaten Prüfregelsätze problemlos weiterverwenden. CROCODILE liefert RAT-Prüfergebnisse sowohl im klassischen, RAT-konformen Format, also auch in CROCODILE-typischer, logisch vernetzter Hypertext-Darstellung — einschließlich einer vergleichenden Gegenüberstellung der Ergebnisse von Wiederholungsprüfungen (mehr dazu in Kapitel 7.2 im Abschnitt «Aufbereitung einer Vergleichsdarstellung von Evaluationsergebnissen» auf Seite 111). RAT-Prüfergebnisse werden in den CROCODILE-Prüfreport an passender Stelle eingefügt, und das Prüfergebnis wird um RAT-spezifische Hinweise, Korrekturvorschläge und Querverweise ergänzt.

13 Streng genommen besteht die Möglichkeit, statt eines Patterns eine in PERL zu schreibende Prüfroutine — ein sogenanntes "Callout" — bereitzustellen, das dann die IOS-Konfigurationsbeschreibung beliebigen Tests unterziehen kann. RAT bietet jedoch kaum Unterstützung für Callouts und Callout-Programmierung, und das Prüfergebnis eines Callouts ist auf ein simples 'PASS' oder 'FAIL' beschränkt. Es handelt sich also eher um einen Notanker, um die strikten funktionalen Beschränkungen des RAT-Ansatzes im Einzelfall zu überwinden. Die Standarddistribution RATv2.0 vom März 2003 enthält nur vier recht triviale Callouts: CheckBufferingSizeMin(), CheckIOSPasswordQuality(), MatchAtLeast(), CheckIOSSecretReuse().

Copyright © Fraunhofer IESE 2003

5.2 RAT-Konformität

Bei der Entwicklung des Emulationsmoduls waren wir bemüht, alle Prüfschritte in einer zu RAT äquivalenten Form durchzuführen. Aufgrund von Urheber- und Lizenzrechten dürfen jedoch nur CIS-akkreditierte Werkzeuge oder -Auditoren eine CIS-Benchmark-Wertung vergeben, und CROCODILE ist derzeit nicht akkreditiert. Die Verarbeitung von RAT-Prüfregelsätzen erfolgt daher ohne Gewähr und ohne den förmlichen Anspruch irgendeiner CIS-Konformität. Der Anwender mag anhand unserer Prüfreports selbst entscheiden, wie er die Originaltreue des Emulators beurteilt und welchen Stellenwert er dem Prüfergebnis unserer RAT-Emulation beimißt. Wir nehmen für CROCODILE in Anspruch, daß es RAT-Prüfregelsätze getreulich anwendet.

In einigen Details weicht die Regelverarbeitungen unter CROCODILE allerdings erkennbar von der unter RAT ab. Geringfügige Unterschiede ergaben sich vor allem durch Schwächen der RAT-Software. So fanden sich bei der Analyse der RAT-Distribution (Stand März 2003) zum Beispiel

- *Tippfehler* in den Regelspezifikationen, die RAT ohne Warnung ignoriert, die jedoch Einfluß auf die Regelanwendung haben können;
- *Nicht vorgesehene Matches* einzelner Patterns, die unter RAT in Testbeispielen eine unzulässige Regelanwendung hervorriefen;
- *Verfehlte Matches von korrekt spezifizierten Patterns,* die unter RAT in Testbeispielen eine anwendbare Regel unbeachtet ließen;
- *Inkonsistenzen* wie etwa Querverweise auf undefinierte Regelobjekte, die RAT unbeanstandet ließ und offenbar gar nicht bemerkte.

Die vorgefundenen Mängel haben wir dem RAT Entwicklerteam mitgeteilt. Es ist davon auszugehen, daß diese Problemfälle mittlerweise beseitigt worden sind. Das Emulationsmodul wurde auf alle Fälle so gestaltet, daß es die Fehler von RAT vermeidet und Patterns korrekt anwendet, offensichtlich falsche Patterns zum Teil sogar automatisch korrigiert und erkennbare Fehler und Inkonsistenzen in RAT-Regeldaten auf der Konsole zumindest protokolliert. Sofern nach Entdecken einer Regelschwäche ein Weiterarbeiten noch sinnvoll möglich erscheint, setzt CROCODILE den RAT-Prüfschritt trotz Warnmeldung fort und erzielt oft noch ein sinnvolles Prüfergebnis.

Da wir also von den RAT-Mechanismen laut verfügbarem Software-Stand abweichen mußten um RAT-Fehler zuvermeiden, kann dies im Detail kleinere Unterschiede zwischen RAT- und CROCODILE-Prüfergebnissen hervorrufen. So ergibt sich natürlich ein anderes Benchmark-Resultat, wenn RAT fälschlich Regeln ausläßt (oder anwendet), die CROCODILE korrekterweise ausführt (beziehungs-

Copyright © Fraunhofer IESE 2003

weise ignoriert). Daher sind manchmal minimale Diskrepanzen zwischen RAT und CROCODILE möglich. Der Anwender kann sich jedoch leicht davon überzeugen, daß dies normalerweise keinen nennenswerten Einfluß auf das Prüfergebnis hat.

5.3 Konfiguration des Emulationsmoduls

Den genauen Prüfauftrag erhält das Modul über die Konfigurationsdatei 'RATemulation.conf' im Unterverzeichnis './Configuration' (vgl. Verzeichnisstruktur in Abbildung 10, Seite 43). Dort wird festgelegt

- ob überhaupt RAT-Regelsätze angewendet werden sollen,
- unter welchen Pfadnamen die RAT-Regelspezifikationen und zugehörige Regel-Metadaten gespeichert sind,
- wo sich gegebenfalls der Callout-Code (vgl. Fußnote 13 auf Seite 86) findet für Regeln, die anwenderspezifischen Prüfcode einbinden.

Abbildung 22 zeigt einen typischen Inhalt der Konfigurationsdatei. In der Regel kann davon ausgegangen werden, daß RAT-Interessenten ein Exemplar der RAT-Distribution heruntergeladen und installiert haben. In diesem Falle ergeben sich die erforderlichen Angaben ganz natürlich aus der Struktur des RAT-Stammverzeichnisses. Der Anwender kann die geforderten Daten aber auch unter völlig anderen Pfadnamen hinterlegen, etwa im CROCODILE-Konfigurationsverzeichnis.

Folgende Parameter sind zu setzen:

- RATLIB: *<Pfad zum lib-Verzeichnis einer RATv2.0-Distribution>*
  Diese Angabe ist zwingend erforderlich, falls Callout-Regeln verwendet werden sollen, denn einige Callouts verwenden Datendeklarationen der RAT-Standarddistribution; fehlt die Angabe oder ist ein Zugriff nicht möglich, so werden Callout-Regeln später im Prüflauf (mit Warnhinweis!) ausgelassen — konventionelle RAT-Prüfregeln sollten hingegen trotzdem noch funktionieren.
- CONTEXTS: *<Pfad zur Datei mit den IOS-Kontextbeschreibungen>*
  In der RAT-Standarddistribution finden sich die Kontext-Spezifikationen für IOS gewöhnlich im Unterverzeichnis 'etc/configs/cisco-ios' in der Datei 'contexts.txt'. Hier wird festgelegt, welche Analysekontexte RAT unterscheiden kann und wie diese benannt werden.
- FIELDS: *<Pfad zur Datei mit den Feldformatbeschreibungen>*
  In der RAT-Standarddistribution finden sich die Feldformatbeschreibungen

Copyright © Fraunhofer IESE 2003

```
# Should the RATemulation module be started at all?

  ACTIVE = YES

# Where is the RAT library (required for Callout execution)?

  RATLIB = /rat2.0/lib

# Where is the specification of IOS contexts?

  CONTEXTS = /rat2.0/etc/configs/cisco-ios/contexts.txt

# Where is the specification of valid rule formats?

  FIELDS = /rat2.0/etc/configs/cisco-ios/fields.txt

# Where are the specifications for RAT Globals, RAT Parameters,
# and RAT Evaluation Rules?

  RULES = /rat2.0/etc/configs/cisco-ios/common.conf,
          /rat2.0/etc/configs/cisco-ios/cis-level-1.conf,
          /rat2.0/etc/configs/cisco-ios/cis-level-2.conf,
          /home/schwarz/rat/my-wonderful_private_ruleset.conf

# Where is the NSA Router Security Configuration Guide
# (to be hyperlinked to RAT rules)

  SECURITY_GUIDE = /rat2.0/doc/rscg.pdf

# Do you prefer the original RAT scoring method (which is rather
# obscure)?

  WEIRD_RAT_SCORING = YES

# Should we add page labels to PDF hrefs? This is convenient but
# does not work for all browsers.

  PDF_PAGE_LABELS = NO
```

Abbildung 22 Beispiel für Vorgaben in 'RATemulation.conf'

gewöhnlich im Unterverzeichnis 'etc/configs/cisco-ios' in der Datei 'fields.txt'. Die Feldformatbeschreibungen legen fest, welche Attribute in RAT-Regeln vorgeschrieben oder zulässig sind und welchen Wertebereich diese Attribute annehmen dürfen.

- RULES: <Liste mit Pfaden zu Prüfregeldateien>
  In der RAT-Standarddistribution finden sich einige Standardregelsätze im

Copyright © Fraunhofer IESE 2003

Unterverzeichnis 'etc/configs/cisco-ios' in den Dateien 'common.conf' (grundlegende Deklaration), 'cis-level-1.conf' (Prüfregeln für elementare Sicherheitsanforderungen) sowie 'cis-level-2.conf' (Prüfregeln für erweiterte Sicherheitsanforderungen). Benutzerspezifische Zusatzregeln erwartet RAT gewöhnlich in 'local.conf' im gleichen Unterverzeichnis. Beliebig viele Pfade zu beliebigen Verzeichnissen dürfen unter 'RULES' aufgelistet werden.

- SECURITY_GUIDE: *<Pfad zum Router Security Configuration Guide>*
  Die RAT-Standarddistribution enthält den Router Security Configuration Guide [NSAguide] unter '/doc/rscg.pdf'. Wird ein ungültiger Pfad angegeben, so kann CROCODILE keine entsprechenden Hyperlinks als Querverweise generieren, wie es laut Standardregelspezifikation vorgesehen ist. Dies hat jedoch keinen Einfluß auf das Prüfergebnis selbst.

- PDF_PAGE_LABELS: 'yes' oder 'no'
  Bei der Angabe 'yes' generiert CROCODILE HREFs auf PDF-Dateien, die auf eine bestimmte Seite innerhalb des PDF-Dokuments verweisen. Das Format dazu lautet '<a href="*<Dateiname>*.pdf#page=*<Seitennummer>*"…'. Leider wird dieses Format nur von bestimmten Browsern und nur mit dem passenden Adobe Plugin unterstützt — manche Browser sind nicht in der Lage, solche HREFS überhaupt zu verstehen. Sollte ein PDF-Link in einem RAT-Report nicht funktionieren, so kann man mit der Angabe 'no' erzwingen, daß alle Page Labels weggelassen werden.

- WEIRD_RAT_SCORING: 'yes' oder 'no'
  Bei der Angabe 'yes' berechnet CROCODILE aus Kompatibilitätsgründen die zusammenfassende Kennzahl des CIS-Benchmarks genau wie die RAT-Software. Unabhängig davon berechnet CROCODILE noch eine eigene Kennzahl nach modifizierter Vorschrift, da die Originalformel fragwürdig erscheint (siehe Abbildung 23 auf Seite 92): Mehr dazu im folgenden Abschnitt.

5.4 Kennzahlberechnung (CIS Benchmark und CROCODILE Score)

Das RAT-Framework realisiert in seiner Gesamtheit den von CIS vorgeschlagenen Sicherheits-Benchmark für IOS Router. Zur einfachen Vergleichbarkeit wird die Gesamtheit aller Prüfbefunde zu einer einzelnen Kennzahl aus dem Intervall [0 … 10] verdichtet. Je näher das Gesamtergebnis am Idealwert 10.0 liegt, als um so sicherer ist die Routerkonfiguration einzustufen.

Leider liegt der Berechnung der Kennzahl folgende, recht fragwürdige Rechenvorschrift zugrunde:

1. Jede Prüfregel wird auf alle IOS-Klauseln angewendet, die dem Regel-

Copyright © Fraunhofer IESE 2003

Pattern entsprechen. Jede Regelanwendung gilt als ein sogenannter «check».

2. Jeder «check» mit dem Ergebnis 'FAIL' wird unter der Kategorie «checks failed» gezählt.

3. Wenn eine Prüfregel mehrmals angewendet wird und einige «checks» 'FAIL', andere hingegen 'PASS' liefern, so bleiben die erfolgreichen Regelanwendungen unberücksichtigt: Nur die «checks» mit dem Ergebnis 'FAIL' werden unter der Kategorie «checks failed» gezählt.

4. Wenn eine Prüfregel ein- oder mehrmals angewendet wird und sämtliche «checks» das Ergebnis 'PASS' liefern, so wird dies dennoch nur als ein einziger erfolgreicher «check» unter der Kategorie «checks passed» gezählt. Wiederholter Erfolg der gleichen Regel bleibt in jedem Falle also unberücksichtigt.

5. Das Verhältnis der Summe der Gewichte (Regelattribut 'Importance') aller «checks passed» zur Summe der Gewichte aller «checks failed» und «checks passed» liefert den gesuchten Wert des Benchmarks.

Die Berechnungsschritte 3 und 4 betonen offensichtlich Fehlschläge gegenüber Erfolgen einer Regelanwendung. Dies gibt mitunter ein recht verzerrtes Bild: Wenn etwa von 10 Interfaces neun korrekt und nur eines fehlerhaft bezüglich einer Regel konfiguriert wurden, so vermeldet der RAT-Prüfreport genau eine fehlgeschlagene Regelanwendung — die Masse der erfolgreichen Regelanwendungen wird nicht gezählt, ja nicht einmal protokolliert! Wäre dies die einzige Regel, die im Prüflauf zur Anwendung kommt, so ergäbe sich folglich ein RAT-Score von 0.0, obwohl die Konfiguration beinahe korrekt ist.

Zur Vermeidung solcher Zerreffekte liefert CROCODILE zusätzlich zur Original-Kennzahl nach CIS-Vorschrift noch die entsprechende Kennzahl, die sich durch gleichberechtigte Zählung *aller* fehlgeschlagenen und erfolgreichen «checks» ergibt. Anders als unter RAT werden von CROCODILE außerdem auch alle «checks» protokolliert, jedes 'PASS' und jedes 'FAIL'. Das CROCODILE-Prüfprotokoll ist somit vollständiger, und der CROCODILE-Score ist im allgemeinen etwas höher als sein CIS-Pendant.

5.5 Prüfreports

Das Emulationsmodul liefert die einschlägigen Prüfberichte in RAT-konformem Format, so daß der Umstieg von RAT auf CROCODILE dem Anwender keine Probleme bereiten sollte. Abbildung 23 zeigt das typische Format eines solchen RAT-Reports. Zeilen, die sich auf bestimmte Prüfregeln beziehen, sind per Hypertext-Link mit der zugehörigen Regelbeschreibung einschließlich zugeord-

Copyright © Fraunhofer IESE 2003

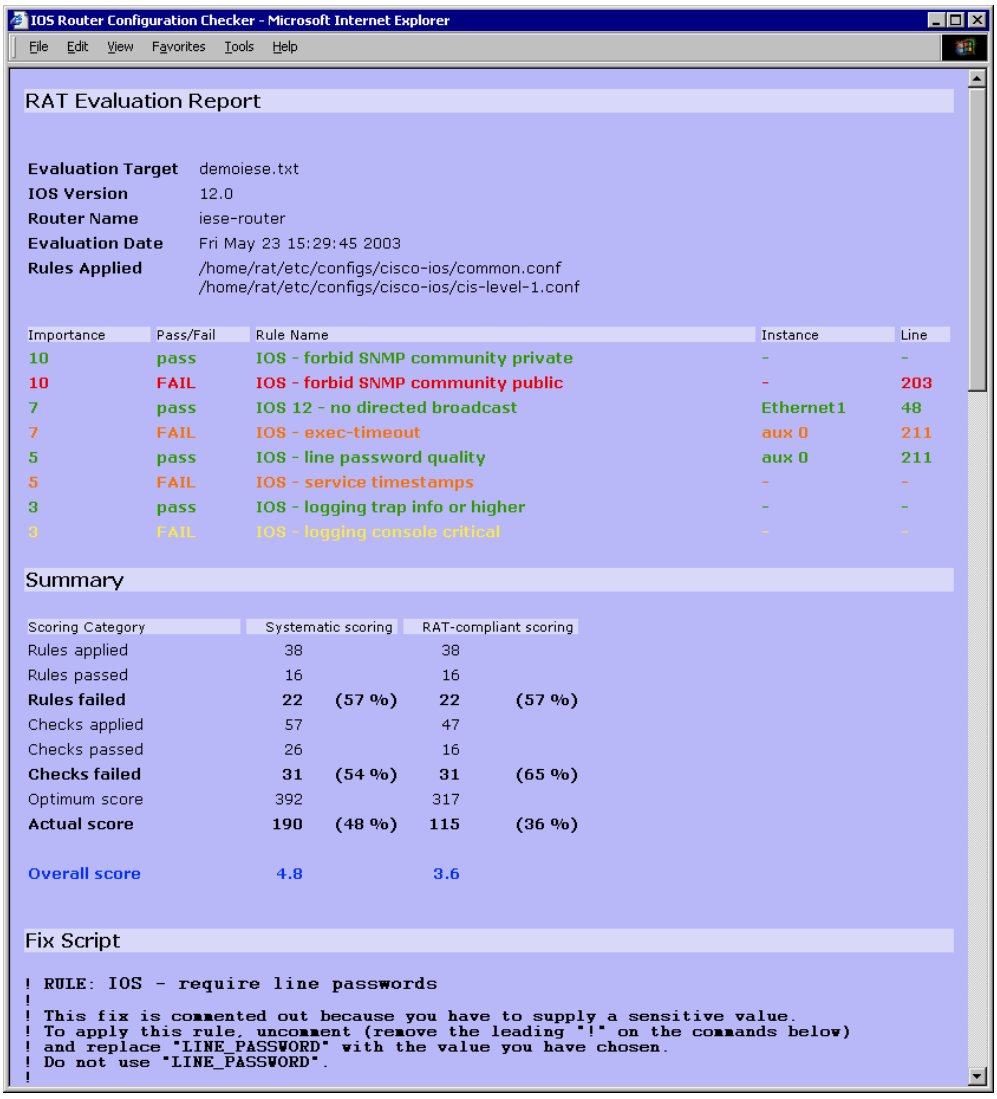

Abbildung 23 Ausschnitt aus einem typischen, von CROCODILE generierten RAT-Prüfreports

neter Querverweise verbunden. Per Mausklick kann man diese Hintergrundinformationen in die Anzeige holen.

Eine Zusammenfassung der RAT-spezifischen Ergebnisdaten wird unter dem Bookmark «RAT Emulation» im Hypertext-Prüfreport bereitgestellt. Zusätzlich ordnet CROCODILE jeden einzelnen RAT-Befund den verschiedenen Anzeigen und Kategorien der CROCODILE-Darstellung zu. RAT-Ergebnisse werden also genau wie gewöhnliche CROCODILE-Befunde behandelt. Ist zum Beispiel eine Zeile, ein Klauselkontext oder ein logischen Kontext von einem RAT-Befund berührt, so

Copyright © Fraunhofer IESE 2003

wird der Befund in die betroffenen Zeilen-, Klausel- oder Kontextanzeigen einschließlich der «Overview»-Darstellung eingeblendet. Zudem wird für das RAT-Modul ein zusätzlicher Profil-Balken im «Evaluation Profile» angelegt.

Die doppelte Ergebnisaufbereitung sowohl nach RAT- als auch nach CROCODILE-Konventionen liefert allen Anwendern — auch ehemaligen RAT-Nutzern — den jeweils vertrauten Prüfreport.

5.6 Regelbeschreibungen

RAT-Prüfregeln lassen sich hierarchisch in Klassen und Unterklassen untergliedern. Eine RAT-Regelklasse entspricht in etwa einem «Prüfbereich» im CROCODILE-Sprachgebrauch. Die Baumstruktur der Regelhierarchie ermöglicht es, ganze Teilbäume geschlossen auszuwählen oder zu deaktivieren. Dazu hat jede Regelklassen- und Regelbeschreibung ein Attribut 'Selected' mit den möglichen Attributwerten 'YES' oder 'NO'.

Das Original-RAT beinhaltet eigene Editierwerkzeuge für Regeldatensätze, mit denen sich einzelne Prüfbereiche komfortabel auswählen und aktivieren lassen. Der Regeleditor sorgt dafür, daß die entsprechenden Attribute korrekt und konsistent gesetzt werden.

Etliche Prüfregeln sind auch mit Variablen parametrisierbar. So lassen sich Regeln zum Beispiel an lokal vergebene Netzwerkadressen anpassen. Die zugehörigen Variablenbelegungen sind Teil der Prüfregelspezifikation und müssen vor einem RAT-Prüflauf vom Anwender angepaßt werden. Auch dabei unterstützt der RAT-Regeleditor den Anwender.

CROCODILE erkennt sowohl Regelklassen und Regelhierarchien als auch konfigurierbare Regelparameter. Es überprüft deren Stimmigkeit und wendet die definierten Einstellungen korrekt an. Wird zum Beispiel ein Regelklasse mittels «Selected: no» deaktiviert, so werden automatisch alle zu dieser Klasse gehörenden Regeln ignoriert, selbst wenn einzelne Regeln der Klasse individuell mit «Selected: yes» aktiviert sind.

Unter dem Bookmark «RAT Emulation» findet sich der Link «Rule Descriptions», der auf eine Gesamtbeschreibung aller konfigurierten Regelspezifikationen verweist. Die Beschreibung ist als Hypertext-Dokument aufbereitet und enthält sämtliche Regelklassen, Regeln und Datenparameter mit ihren wichtigsten Attributen. Die Regelhierarchie wird durch Hyperlinks repräsentiert, mit denen die Klassen- und Regelnamen hinterlegt sind. Dies erlaubt das bequeme Navigieren im Prüfregelbaum. Deaktivierte Teile der Regelspezifikation sind durch Graufärbung kenntlich gemacht.

Copyright © Fraunhofer IESE 2003

CROCODILE verzichtet allerdings auf jegliche Editor-Unterstützung für RAT-Prüfregelsätze: Anwender haben meist ohnehin einen RAT-Editor aus der Originaldistribution zur Hand, und notfalls können Prüfregeln auch recht einfach — wenn auch weniger komfortabel — mit einem gewöhnlichen Texteditor bearbeitet werden.

Für nähere Hinweise zum genauen Format der Regelbeschreibungen und den Einstellungsmöglichkeiten verweisen wir auf die Dokumentation der Originaldistribution [RAT2.0].

5.7 Bewertung

Das RAT-Emulationsmoduls ist in mehrerlei Hinsicht eine Bereicherung für CROCODILE:

- Es ermöglicht RAT-Anwendern den einfachen Umstieg auf CROCODILE, wobei Investitionen in die Definition eigener RAT-Prüfregelsätze gewahrt bleiben.
- Es bereichert den CROCODILE-Hypertext-Report um weitere Querverweise und Korrekturvorschläge.
- Es sichert die Teilhabe an Prüfregeln, Callouts, Fix Suggestions und Literaturhinweisen, die vom CIS und anderen künftig noch im RAT-Format veröffentlich werden.

Umgekehrt kann CROCODILE sogar eine Bereicherung für RAT sein, denn es vermeidet oder kompensiert einige der Unzulänglichkeiten der von uns zugrunde gelegten RAT-Distribution und liefert bei gleichen Prüfregeln-Vorgaben reichhaltigere Prüfberichte.

Aus CROCODILE-Sicht stehen dem Anwender nunmehr Spezifikationsmöglichkeiten auf unterschiedlichen Niveaus zur Verfügung:

- Simple und offensichtliche Prüfkriterien, die keiner weiteren Erläuterung bedürfen, lassen sich am schnellsten und bequemsten mittels 'Compound-Patterns' konfigurieren. Mittels Compounds lassen sich dabei auch Regeln für mehrere Konfigurationsklauseln logisch verbinden; auch bedingte Regeln sind so formulierbar.
- Prüfkriterien, die zwar einfach aber weniger offensichtlich sind und weiterer Erläuterungen bedürfen, lassen sie sich am besten im RAT-Format konfigurieren. Der Vorteil besteht darin, daß auf diesem Wege Querverweise, Erklärungen und Korrekturvorschläge ohne Programmieraufwand in den Prüfbericht eingefügt werden können — der Nachteil besteht vor allem im

Copyright © Fraunhofer IESE 2003

vergleichsweise hohen Spezifikationsaufwand und der gegenüber 'CompoundPatterns' geringeren Ausdruckskraft .

- Das Spezifikationsformat für 'IngressEgress' ist speziell an die Bedürfnisse von ACL-Prüfvorgaben angepaßt und eröffnet Möglichkeiten für Prüfregeln, die weder 'CompoundPatterns' noch 'RATemulation' auch nur annähernd zu bieten vermögen.

- Für Prüfkriterien, die sich mit keinem der verfügbaren Konfigurationsformate darstellen lassen, bietet das Framework eine recht mächtige Programmierschnittstelle, um zusätzliche Prüfmodule zu implementieren. Solche Module sind möglicherweise konfigurierbar in einem auf den Prüfbereich zugeschnittenen Format.

Programmierung neuer Prüfmodule bleibt für den gewöhnlichen Anwender die Ultima Ratio: In vielen Fällen sind die vorhandenen Konfigurationsformate ausreichend, um die gewünschten Prüfkriterien zu formulieren. Wir rechnen damit, daß es in Zukunft noch weitere, flexibel konfigurierbare Prüfmodule für CROCODILE geben wird. Einstweilen bieten 'IngressEgress', 'CompoundPatterns' und 'RATemulation' interessante Möglichkeiten zur Abwägungen zwischen Universalität, Ausdrucksstärke und Bequemlichkeit des Spezifikationsformats.

Copyright © Fraunhofer IESE 2003

Copyright © Fraunhofer IESE 2003

6 Stapelverarbeitung (Batch Processing)

CROCODILE ist in seinen Ursprüngen ein interaktives Werkzeug, um eine einzelne IOS-Konfiguration einer genauen Prüfung zu unterziehen. Betreiber größerer Netzwerke, die viele IOS-Router zu überwachen haben, müssen mitunter jedoch flächendeckende Reihenuntersuchungen durchführen. Dies erfordert den Stapelverarbeitungsbetrieb im unbetreuten OFFLINE-Modus.

CROCODILE unterstützt Batch Processing durch ein separates Frontend-Programm: 'batchcroco'. Im folgenden wird der Stapelverarbeitungsmodus des Werkzeugs dargestellt und erläutert.

6.1 Verwendung von CROCODILE im Stapelverarbeitungsmodus

Der Stapelverarbeitungsmodus dient dazu, mehrere IOS-Konfigurationsdateien in Serie zu analysieren, pro Datei einen Prüfbericht zu erstellen und die Prüfbefunde für alle Konfigurationsdateien tabellarisch aufzubereiten.

Die Stapelverarbeitung übernimmt das Programm 'batchcroco' im CROCODILE Hauptverzeichnis. Das Programm wird mit den Namen der zu untersuchenden Dateien als Aufrufparameter gestartet. Die Namensliste darf auch Verzeichnisnamen enthalten: In diesem Falle werden alle Dateien im angegebenen Verzeichnis der Stapelverarbeitung unterworfen. Auf Wunsch werden dabei auch alle Unterverzeichnisse rekursiv durchsucht und in die Verarbeitung mit einbezogen.

Achtung: *Je nach Betriebssystem dürfen Dateinamen auch Leerzeichen enthalten. Damit Datei-Listen als Kommandozeilen-Parameter eindeutig interpretierbar sind, müssen solche Dateinamen — oder zumindest die darin enthaltenen Leerzeichen — nach den Konventionen der Plattform geeignet in «Quotes» gesetzt werden. Ob dies mit einfachen oder doppelten Anführungsstriche oder zum Beispiel mit* '\' *zu erfolgen hat, hängt vom Betriebssystem ab.*

Als Argument sind auch Muster mit sogenannten Wildcard-Zeichen ('*' bzw. '?') zulässig. Typische Unix-Shells lösen solche (und noch komplexere) Suchmuster bereits auf Shell-Ebene auf und übergeben der Anwendung die dem Muster entsprechende, expandierte Datei-Liste. Windows-Konsolen reichen Wildcards dagegen an die Anwendung durch. Wenn CROCODILE Wildcards in Dateinamen

Copyright © Fraunhofer IESE 2003

entdeckt, die noch nicht auf Shell-Ebene aufgelöst wurden, so interpretiert es diese als Suchmuster und expandiert sie auf Anwendungsebene.

Achtung: *Je nach Betriebssystem wird Groß/Kleinschreibung bei Dateinamen beachtet oder ignoriert, mitunter auch uneinheitlich je nach Anwendung. Die Auflösung von Suchmustern liefert daher mal alle «Treffer» unabhängig von ihrer Schreibung, mal nur solche Dateinamen mit übereinstimmender Groß/Kleinschreibung. Der Anwender sollte sich auf seiner jeweiligen Plattform vergewissern, daß Wildcards seinen Erwartungen gemäß aufgelöst werden.*

Um etwa alle Dateien mit der Endung '.txt' im Verzeichnis 'SampleConfigs' zu analysieren, lautet der Aufruf unter einem Unix-Betriebssystem

```
user@host[CROCODILE]  perl batchcroco SampleConfigs/*.txt

CROCODILE batch processing started.

=====================================================================
=                                                                   =
=  Processing 20 batch jobs        2003/09/25   14:16:08           =
=                                                                   =
=====================================================================
--- Job 1  of 20: SampleConfigs/acsac.txt -------------------------
--- Job 2  of 20: SampleConfigs/brian1.txt ------------------------
--- Job 3  of 20: SampleConfigs/brian2.txt ------------------------
…
--- Job 19 of 20: SampleConfigs/sft.txt ---------------------------
--- Job 20 of 20: SampleConfigs/some1.txt -------------------------

Writing summary to '/tmp/batchresults/summary.html' ...
FILENAME     FINDINGS   OKAYS INFOS CHCKS WARNS ALRTS  PROBMS  ERRS
ibmt.txt          165      40    31    31    51    12
acsac.txt         168      65    30    27    30    16
ict.txt           167      49    28    29    50    11
ciosc.txt         168      70    29    26    30    13
…
ise.pix.txt       161      23    69    18    41    10       6
some1.txt         156      54    24    29    39    10
brian1.txt        189      70    36    33    41     9
brian2.txt        185      63    42    30    41     9
sft.txt           103      16    22    27    34     4

Writing all findings to '/tmp/batchresults/allfindings.txt' ...

Batch process terminated.

Batch jobs with PROBLEMS: 1  ERRORS: 0  ABORTS: 0 INTERRUPTS: 0
Overall user time 13:35 min   system time 0:12 min
```


Copyright © Fraunhofer IESE 2003

Das Programm schreibt die Ergebnisdateien in ein vom Benutzer konfigurierbares Verzeichnis, im Beispiel oben ist dies '/tmp/batchresults'. Das Ergebnis der Stapelverarbeitung umfaßt

- ein Protokoll des gesamten Stapelverarbeitungslaufs: 'batchprocess.log';
- eine Tabelle mit den Einzelergebnissen aller Jobs: 'summary.<format>';
- einen Prüfbericht pro Eingabedatei: '<file name>.<format>';
- gegebenenfalls eine Liste aller Befunde aller Jobs in maschinenlesbarem Textformat: 'allfindings.txt'.

Die Dateiendung richtet sich bei der Gesamttabelle und den Prüfberichten nach dem gewünschten Ausgabeformat — entweder ASCII-Text (Endung 'txt') oder HTML (Endung 'html'). Die Datei 'allfindings.txt' wird, falls gewünscht, nur im Textformat erzeugt, eine HTML-Version ist nicht vorgesehen.

Abbildung 24 zeigt zwei typische Beispiele solcher Ergebnisdateien. Die Einträge in 'summary.html' (oben rechts) sind als Hyperlinks ausgelegt. Durch Anklicken einer Tabellenzeile läßt sich sofort der zugehörige Prüfbericht mit den detaillierten Befunden öffnen (in Abbildung 24 unten links im Hintergrund dargestellt). Der im Batch-Modus generierte HTML-Prüfbericht ist identisch mit dem 'Evaluation Report', den 'crocodile' im interaktiven Modus erzeugt. Darüber hinaus erzeugt 'crocodile' jedoch weitere Sichten und Zusatzinformationen, die im Stapelverarbeitungsbetrieb mit 'batchcroco' nicht generiert werden.

Prüfergebnisse im ASCII-Format werden nur im Stapelverarbeitungsmodus erzeugt. Im interaktiven Modus ist ausschließlich HTML bzw. XML verfügbar.

Fehlschlag oder Abbruch einzelner Batch Jobs

Das Programm 'batchcroco' ist lediglich ein Frontend, das die einzelnen Jobs mit geeigneten Aufrufparametern letztlich an 'crocodile' zur Ausführung übergibt. Ein einzelner Batch Job kann Warn- oder Fehlermeldungen hervorrufen. Solche «besonderen Vorkommnisse» werden in der zusammenfassenden Darstellung vermerkt und in der Logdatei 'batchprocess.log' erfaßt (siehe etwa 'ise.pix.txt' in Abbildung 24: Hier sind 6 «Probleme» aufgetreten). Die Abschlußmeldung des Stapelverarbeitungslaufs nennt die Zahl der aufgetretenen Probleme und Fehler.

Ein Fehler kann so schwerwiegend sein, daß eine Fortsetzung des Jobs nicht mehr sinnvoll möglich ist. In diesem Falle wird der betroffene Job kontrolliert abgebrochen, und in der Zusammenfassung wird der Job als «aborted» gekennzeichnet. Ein Befund wird für diesen Job nicht ausgewiesen. Der Stapelverarbei-

Copyright © Fraunhofer IESE 2003

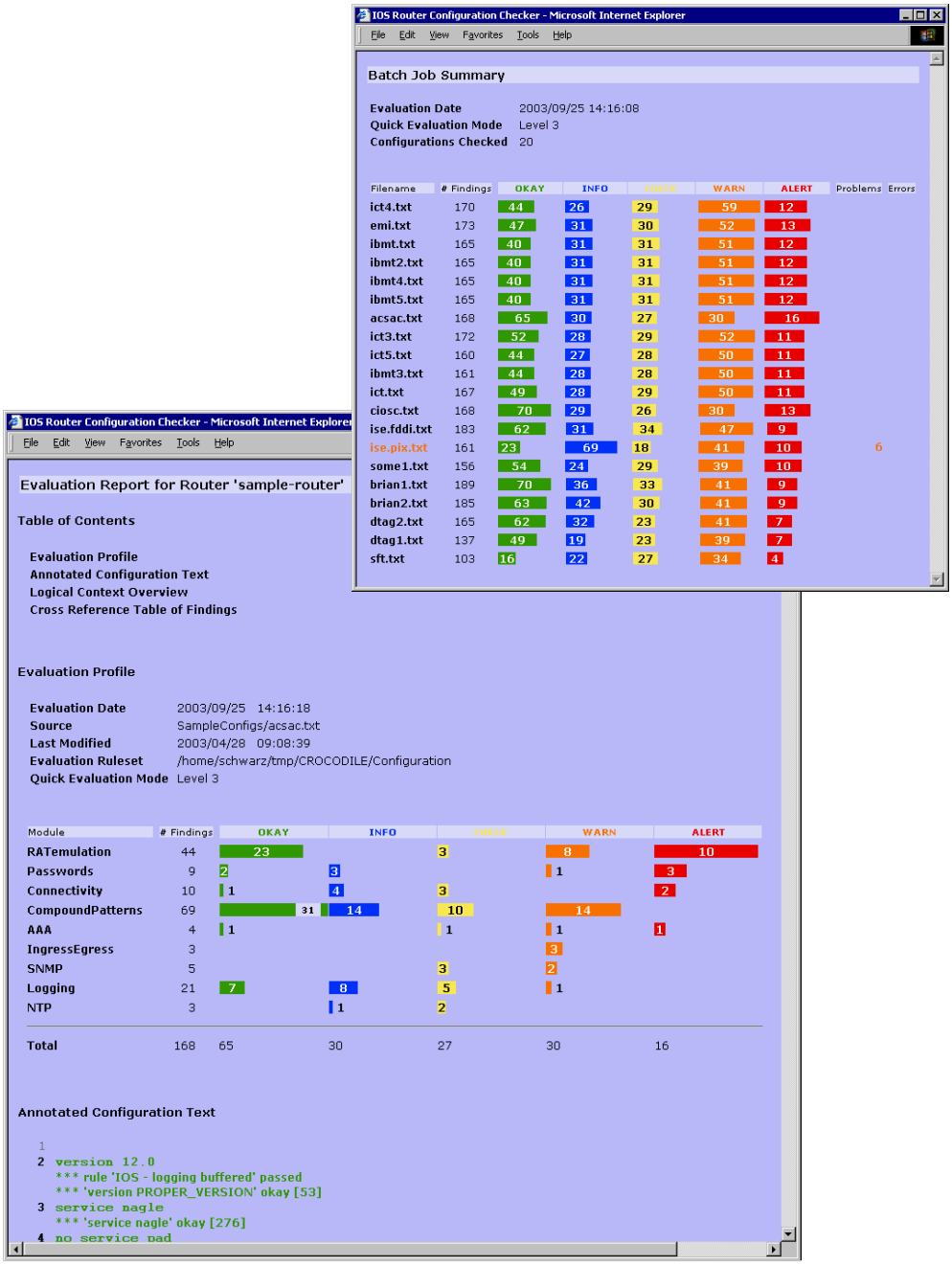

Abbildung 24 HTML-Ergebnisse eines Stapelverarbeitungslaufs: 'summary.html' und 'acsac.html'

tungsbetrieb wird jedoch mit dem nächsten Job normal fortgesetzt. CROCODILE weist die Anzahl der abgebrochenen Jobs in einer Abschlußmeldung auf der Konsole aus.

Copyright © Fraunhofer IESE 2003

Benutzergesteuerte Jobunterbrechung

Es besteht auch die Möglichkeit, den gerade laufenden Job interaktiv mittels der Tastenkombination 'Control-C' abzubrechen. Der Job wird dann ohne Befund beendet und in der Zusammenfassung als «interrupted» gekennzeichnet. Das Werkzeug fragt daraufhin ab, ob mit dem nächsten Job fortgefahren werden soll. Sofern der Anwender diese Frage nicht ausdrücklich mit 'n' verneint, wird der Batch-Prozeß mit dem nächsten noch ausstehenden Job fortgesetzt.

Die Unterbrechung mit 'Control-C' bietet die Möglichkeit, einzelne Jobs zu überspringen, ohne daß dadurch die bereits vorliegenden Prüfergebnisse verloren gehen. Die Anzahl der unterbrochenen Jobs wird in der Abschlußmeldung auf der Konsole ausgewiesen.

Achtung: *Dieses Leistungsmerkmal funktioniert derzeit leider nicht unter allen Betriebssystemen. Wird CROCODILE unter Microsoft Windows in einer Shell gestartet, so leitet Windows die Unterbrechungssignale bei mehrfachen Unterbrechungen nicht ordnungsgemäß an die Signalverarbeitung des Perl-Laufzeitsystems weiter. Das kontrollierte Abfangen der zweiten und folgender 'Control C' Unterbrechungen entzieht sich daher unter Windows dem Einfluß von CROCODILE.*

6.2 Konfigurierbare Ausführungsparameter (Options)

Grundsätzlich kann der Anwender 'batchcroco' ohne die Angabe besonderer Aufrufoptionen verwenden. Das Programm arbeitet mit Voreinstellungen, die in der Regel automatisch für brauchbare Ergebnisse sorgen.

Diese Voreinstellungen können wahlweise aber auch geändert werden, indem man die gewünschten Parameterwerte in der Datei './Configuration/batch.conf' einträgt. Parameter, die in der Konfigurationsdatei nicht ausdrücklich gesetzt werden, sind automatisch mit Vorgabewerten belegt. In Tabelle 3 sind alle konfigurierbaren Optionen und deren Voreinstellungen aufgelistet.

Alle Einstellungen, die mittels 'batch.conf' möglich sind, kann der Benutzer auch durch Kommandozeilen-Optionen für einen einzelnen Aufruf individuell einstellen. Jede Kommandozeilen-Angabe überschreibt dabei die entsprechende Vorgabe in 'batch.conf'. Tabelle 3 zeigt die Entsprechungen von Konfigurationsparametern und Kommandozeilen-Optionen.

Als zusätzliche Kommandozeilen-Option unterstützt 'batchcroco' den Aufruf der Online-Hilfe mit '-h'. Diese Option ist aus naheliegenden Gründen nicht vorkonfigurierbar.

Copyright © Fraunhofer IESE 2003

Option in 'batch.conf'	Kommando-zeile	Bedeutung	Default
ResultDir	-d	Verzeichnisname, in dem alle Ergebnisdateien abgelegt werden	Current Working Directory
OutputFormat	-f	Formats der Ergebnisdateien: *HTML* oder *ASCII*	*ASCII*
FieldSeparator	-fs	Trennsymbol für Felder in Ausgabedateien im ASCII-Format: *s* (= *SEMICOLON*), *t* (= *TABULATOR*) oder "*<zeichenkette>*"	*TABULATOR*
IncludeSubdirs	-i	Beziehe rekursiv alle Unterverzeichnisse eines Eingabeverzeichnisses in die Verarbeitung ein: *on* oder *off*	*off*
ResultOverwrite	-o	Existierende Ergebnisdateien ohne Warnung überschreiben: *on* oder *off*	*off*
ResultPrefix	-p	Angegebe Zeichenkette im Format <Prefix>- allen Ausgabedateinamen voranstellen	– kein Präfix –
Quick	-q	Überspringen einiger nicht-essentieller Prüfungen im Batch-Modus: *0* (= *off*), *1*, *2* oder *3* (= *on*)	*on*
QuoteFields	-qf	Einschließen aller Textfelder bei Ausgabe im ASCII-Format in doppelte Anführungszeichen: *on* oder *off*	*off*
Ruleset	-r	Verzeichnisname des Prüfregelsatzes (im Oberverzeichnis RulesetDir), der auf die Batch Jobs anzuwenden ist	abgeleitet aus dem Namenspräfix der Eingabedatei "*<set>*-<filename>"
RulesetDir	-s	Verzeichnisname, in dem die Prüfregelsätze (als Unterverzeichnisse) abgelegt sind	Current Working Directory
ResultAddDate	-t	Datums im Format -JJJJMMTT an den Dateinamen aller Ausgabedateien anfügen: *on* oder *off*	*on*
Verbose	-v	Detaillierungsgrad der Protokollierung auf der Konsole: *0* (= *off*), *1* oder *2*(= *on*)	*off*

Tabelle 3 Verfügbare Optionen in 'batch.conf' und deren Entsprechung als Kommandozeilen-Option.

Copyright © Fraunhofer IESE 2003

6.3 Auswahl von Prüfregelsätzen

Je nach Einsatzzweck eines Routers muß dieser unterschiedlich konfiguriert werden. Folglich variieren auch die Prüfkriterien von Fall zu Fall: Was für einen Backbone-Router gut und wichtig ist, mag für einen Access-Router ein klarer Verstoß gegen übliche Sicherheitsrichtlinien sein — und umgekehrt! Dies macht es erforderlich, bei der Stapelverarbeitung einzelne Jobs nach wechselnden Prüfkriterien durchzuführen.

CROCODILE verwaltet seine Prüfregeln in Unterverzeichnissen. In einem Unterverzeichnis ist den verschiedenen Prüfmodulen jeweils eine eigene Konfigurationsdatei zugeordnet. Das heißt, jedes Prüfmodul verwaltet seine Prüfkriterien individuell, unabhängig von allen anderen Modulen. Dem Prüfmodul '<modul>' ist typischerweise eine Konfigurationsdatei '<modul>.conf' im aktiven Prüfregelverzeichnis zugeordnet. Außerdem legt die Datei 'plugin.conf' fest, welche Prüfmodule in einem Prüflauf überhaupt aktiviert werden sollen. Prüfregeln werden somit nach einem Schema gemäß Abbildung 25 verwaltet. Das erleichtert die Wartung der Software und der Prüfregelsätze.

Per Voreinstellung verwendet CROCODILE im interaktiven Betrieb den Prüfregelsatz 'Configuration' im Prüfregelverzeichnis '<CROCOHOME>', dem Heimatverzeichnis des Werkzeugs. Diese Voreinstellung kann der Benutzer beliebig ändern, indem er ein anderes Prüfregelverzeichnis und darin ein anderes Prüfregelsatz-Unterverzeichnis konfiguriert. Dazu dient die Option '-c' des interaktiven Frontends 'crocodile'.

Im Stapelverarbeitungsbetrieb mit 'batchcroco' läßt sich in ähnlicher Weise ein beliebiger Prüfregelsatz einstellen, siehe dazu die Optionen <*RulesetDir*> bzw. <*Ruleset*> in Tabelle 3. Als Vorlage für einen Regelsatz kann das Konfigurations-

Abbildung 25 Strukturierung von Prüfregelsätzen in einem Verzeichnisbaum

Copyright © Fraunhofer IESE 2003

verzeichnis './Configuration' dienen. Wird der Prüfregelsatz mittels dieser Optionen festgelegt, so gilt er für alle Batch Jobs gleichermaßen.

Soll aber statt dessen jeder Job mit einem individuellen Prüfregelsatz ausgeführt werden, so ist eine Prüfregelauswahl über den Namens-Präfix der Eingabedateien möglich. Sobald eine IOS-Konfigurationsdatei mit Namen '<*prefix*>- …' verarbeitet werden soll, so extrahiert 'batchcroco' aus dem Dateinamen den Präfix und wählt im Regelverzeichnis <*RulesetDir*> das Unterverzeichnis <*prefix*> als <*Ruleset*> aus. So würde zum Beispiel der Aufruf

```
user@host[CROCODILE]  perl batchcroco MyRules-acsac.txt -s /Rules
```

die Datei 'MyRules-acsac.txt' nach dem Regelsatz '/Rules/MyRules' analysieren — vorausgesetzt natürlich, daß das Konfigurationsattribut <*Ruleset*> bzw. die Option '-r' nicht gesetzt sind. Wurde ausdrücklich ein Ruleset vorkonfiguriert, so wird die Präfix-Angabe in Dateinamen grundsätzlich ignoriert.

Achtung: *Wenn im Stapelverarbeitungsbetrieb Dateien verarbeitet werden, deren Namen zufällig einen Bindestrich '–' enthält, so wir der Namensbestandteil vor dem Bindestrich als Regelsatz-Angabe interpretiert, sofern das Konfigurationsattribut <Ruleset> nicht spezifiziert ist. Nicht immer liegt dies in der Absicht des Benutzers. Sofern sich nicht zufällig ein Ruleset mit diesem falschen «Regelsatz-Namen» unter <RulesetDir> findet, wird der Job abgebrochen, und es erscheint folgende Fehlermeldung auf der Konsole:*

```
--- Job 1 of 1: SampleConfigs/b-gw12.txt --------------
*** ERROR: Can't find ruleset 'b'.
*** ERROR: File 'SampleConfigs/b-gw12.txt' skipped.
```

Der Pfad zum Regelsatzverzeichnis <*RulesetDir*> kann absolut oder relativ zum Arbeitsverzeichnis des Anwenders angegeben werden. Zusätzlich ist es möglich, den Pfad relativ zum Installationsverzeichnis von CROCODILE anzugeben. Dazu dient das Schlüsselwort '<CROCOHOME>' am Beginn des Pfadnamens.

6.4 Maschinenlesbare Ergebnisformate

Mit der Option '-f ASCII' (bei Bedarf zusätzlich zur Option '-f HTML') kann der Anwender vorgeben, daß 'batchcroco' seine Ergebnisse in gewöhnlichem Textformat ausgibt. Dieses Ausgabeformat ist für einen Menschen weniger gut lesbar als die entsprechende HTML-Ausgabe, es eignet sich aber besser für eine Weiterverarbeitung. Vor allem Unix-Systeme verfügen als «Bordmittel» bereits über zahlreiche Filterwerkzeuge, mit denen Texte schnell und bequem nachverarbeitet werden können. Auch die meisten Textverarbeitungs-, Tabellenkalkulations- und Mathematikprogramme haben in der Regel eine Importschnittstelle für Text.

Copyright © Fraunhofer IESE 2003

CROCODILE erzeugt seine Textausgaben auf Wunsch im «Comma-separated Vector»-Format (CSV), das heißt als Zeilen der Form

```
<Textfeld 1>,<Textfeld 2>,<Textfeld 3>, …
```

Der Trenner zwischen den Textfeldern muß dabei keineswegs immer ein Komma sein. Mittels der Option '-fs' (FieldSeparator) kann auch ein anderes Textzeichen — notfalls sogar eine beliebige Trennphrase — konfiguriert werden. Per Voreinstellung verwendet 'batchcroco' ein Tabulator-Zeichen als Trenner zwischen den Textfeldern. Dieses Format ist auch für Menschen leidlich lesbar; außerdem kann es unmittelbar in Microsoft Excel importiert werden und erzeugt dort ein übersichtliches Tabellenformat. Für den Import in andere Werkzeuge kann es sinnvoll sein, einen anderen Trenner zu wählen, je nach den CSV-Vorlieben des jeweiligen Tools. Übliche Zeichen sind Komma oder Semikolon. Ein anderes Trennsymbol ist insbesondere notwendig, wenn Tabulator-Zeichen auch innerhalb der Textfelder verwendet werden.

Einige Werkzeuge erfordern es, zusätzlich zum Trennzeichen die einzelnen Textfelder in Anführungszeichen einzuschließen. Damit läßt sich meist auch verhindern, daß Sonderzeichen innerhalb der Textfelder die Verarbeitung stören. CROCODILE liefert auf Wunsch solche Anführungszeichen, wenn man die Option 'batchcroco -qf' (QuoteFields) angibt.

Mittels '-f', '-fs' und '-qf' kann der Anwender variabel verschiedene Varianten des CSV-Formats erzeugen. Einer flexiblen Ergebnisaufbereitung (z.B. in Form von Business-Graphiken, Filterung, Aggregation, Sortierung) der Dateien

- summary.txt,
- allfindings.txt

sowie der individuellen ASCII-Prüfreports steht damit nichts mehr im Wege.

6.5 Protokollierung der Stapelverarbeitung

Der Rechenfortschritt des Stapelverarbeitungsprozesses wird auf der Konsole angezeigt. In der Grundkonfiguration ist CROCODILE im Batch-Modus recht «wortkarg» ausgelegt: Die Konsolenausgaben beschränken sich darauf, wesentliche Probleme zu melden und dem Benutzer zu vermitteln, wie weit der Batch-Prozeß vorangeschritten ist.

Sollten irgend welche Verarbeitungsprobleme auftreten, so kann der Benutzer auf Wunsch ausführlichere Konsolenausgaben aktivieren, um den Punkt, an dem die Stapelverarbeitung in Schwierigkeiten geraten ist, möglichst eng einzu-

Copyright © Fraunhofer IESE 2003

grenzen. Dazu dient das Konfigurationsattribut <*Verbose*> (bzw. die Option -v). Es stehen drei Verbosity-Stufen zur Verfügung:

- Stufe 0 (= -v off) — maximale «Einsilbigkeit»
  Auf dieser Stufe werden nur die essentiellen Systemmeldungen des Typs 'error' und 'warning' sowie der Prozeßfortschritt auf der Konsole ausgegeben.

- Stufe 1
  Diese Stufe schließt zusätzlich Systemmeldungen des Typs 'info' in die Konsolenausgabe mit ein.

- Stufe 2 (= -v on) — maximale «Geschwätzigkeit»
  Auf der höchsten Stufe werden sämtliche Systemmeldungen auch auf die Konsole ausgegeben: 'error', 'warning', 'info' und 'verbose'.

Die gewünschte Stufe kann wahlweise als Nummer 0 … 2 oder als Schaltbefehl 'on' bzw. 'off' spezifiziert werden.

Unabhängig von der Belegung des Parameters <*Verbose*> werden alle Systemmeldungen in der Logdatei 'batchprocess.log' protokolliert. Die Datei liegt im konfigurierten Ausgabeverzeichnis <*ResDir*>. Selbst bei sparsamster Konsolenausgabe enthält die Logdatei ein lückenloses Ausführungsprotokoll (Execution Trace) entsprechend Stufe 2. Der Name der Logdatei kann abhängig von den Konfigurationsoptionen <*ResAddDate*> (bzw. '-t') und <*ResPrefix*> (bzw. '-p') um einen vorangestellten Präfix und ein nachgestelltes Datum variieren (vgl. Tabelle 3). Damit sind die Logdaten eindeutig als zu einem bestimmten Prüflauf zugehörig gekennzeichnet.

Copyright © Fraunhofer IESE 2003

7 Hilfswerkzeuge (Utilities)

Im Unterverzeichnis 'Utils' finden sich nützliche Werkzeuge, die den Einsatz des Prüfwerkzeugs unterstützen und sinnvoll ergänzen. Es handelt sich um Funktionen, die während eines Prüflaufs nicht unbedingt benötigt werden. Um Standardanwendungen des Checkers nicht unnötig zu überfrachten, wurden solche Sonderfunktionen in eigenständige Utilities ausgelagert.

7.1 Durchlässigkeitsanalysen an ACLs und Interfaces mittels 'blackwhite'

Mitunter ist es hilfreich genau zu überprüfen, für welche IP-Verkehrsarten eine vorgegebene Access-Control-Liste durchlässig, für welche sie abweisend ist. Wir bezeichnen die Menge aller durchgelassenen IP-Pakete als das «Whiteset» der ACL, die Menge aller abgewiesenen entsprechend als deren «Blackset». Nützlich ist auch die Berechnung aller IP-Pakete, die potentiell über ein bestimmtes Interface des Routers propagiert werden können, das sogenannte «Propagate Set» einer Schnittstelle.

Die Werkzeugumgebung ist in der Lage, eine vollständige Aufstellung aller Whiteset-, Blackset- und Propagate-Set-Paketformate zu generieren. Je nach Umfang und Komplexität der beteiligten ACL ist dazu jedoch ein vergleichsweise hoher Rechenaufwand erforderlich; außerdem können die Mengen in einigen Fällen so groß und «zerklüftet» ausfallen, daß deren Berechnung nur einen geringen Erkenntnisgewinn liefert, der den hohen Interpretationsaufwand für einen menschlichen Benutzer nicht mehr rechtfertigt.[14] Es lag daher nahe, diesen Teil der Analyse in ein eigenständiges Hilfswerkzeug auszulagern — die Utility 'blackwhite'.

Anwendung von 'blackwhite'

Die Verwendung von 'blackwhite' setzt einen Analyselauf mit dem Prüfmodul 'Connectivity' voraus. Ist 'Connectivity' deaktiviert, so fehlen die nötigen Rohdaten für eine nachgeschobene, vertiefende Analyse mittels 'blackwhite'. Der Anwender wird also zunächst einen Standardanalyselauf durchführen, wobei

14 Unglücklicherweise handelt es sich hier um eine ungünstige Rückkopplung: Je länger die Berechnung dauert, um so wahrscheinlicher ist es, daß der Anwender mit dem Ergebnis nur noch wenig anfangen kann.

Copyright © Fraunhofer IESE 2003

'Connectivity' aktiviert sein muß. Daraufhin werden Informationen zu den konfigurierten ACLs im Unterverzeichnis 'XML_data' als XML-Dateien abgelegt.

Ist der Analyselauf des Programms CROCODILE beendet, so kann der Anwender das parameterlose Skript 'Utils/blackwhite.pl' starten. Das Hilfsprogramm findet die unter 'XML_data' abgelegten Daten und bietet dem Anwender ein Menü der zur Auswahl stehenden ACLs und (Line) Interfaces an. Der Anwender wählt interaktiv eine ACL, ein Interface oder ein Line Interface und passende Verarbeitungsoptionen. Das Hilfswerkzeug berechnet daraufhin das Whiteset und Blackset der ACL bzw. das Propagate Set der Schnittstelle entsprechend den Vorgaben. Das Ergebnis wird in gewöhnlichem Textformat nach STDOUT protokolliert. Mit Hilfe von Shell-Funktionen kann der Anwender die Ausgabe bei Bedarf in eine Datei umleiten.

Die Wahl von Restriktionen

Gewöhnlich werden Whiteset, Blackset oder Propagate Set relativ zur Gesamtheit aller möglichen IP-Pakete — also «ip any any» — bestimmt: Für welche Protokolle, Ursprünge, Ursprungs-Ports, Ziele und Ziel-Ports besteht Durchlässigkeit? Eine genaue Antwort läßt sich in der Regel nicht in wenigen Klauseln darstellen. Schlimmstenfalls können die resultierenden Mengen nur als eine lange Liste einzelner Spezialfälle charakterisiert werden.

Fällt das Ergebnis zu umfangreich aus, so ist es nur noch von begrenztem Nutzen, zudem dauert die Berechnung unter Umständen recht lange.[15] Unter diesen Umständen ist es meist sinnvoller, die Durchlässigkeit von ACLs bzw. Schnittstellen relativ zu einer eingeschränkten Ausgangsmenge zu bestimmen, zum Beispiel:

- Welche TCP-Verbindungen, die nicht schon aufgebaut («not established») sind, können passieren?
- Inwieweit ist der Zugriff von Host 6.7.8.9 auf privilegierte Ziel-Ports in den Subnetzen '1.2.3.*' und '5.*.*.*' verwehrt?
- Inwieweit können ICMP 'echo'-Pakete passieren?

Restriktionen dieser Art ermöglichen es dem Anwender, die Durchlässigkeitsanalyse auf beliebig maßgeschneiderte Ausgangsmengen einzuschränken. Dadurch sinkt in der Regel die benötigte Rechenzeit, und die Ergebnisdarstellung fällt verständlicher aus. Sollte sich eine Analyse unter Zugrundelegung von «ip

15 Je nach Leistungsfähigkeit der Hardware und Schwierigkeit der Problemstellung ist mit Rechenzeiten zwischen wenigen Sekunden und bis zu mehreren Minuten zu rechnen. Selbst kniffligste Fälle sollten (bei einem Rechner mit Gigahertz-Taktrate) in weniger als 10 Minuten erledigt sein.

Copyright © Fraunhofer IESE 2003

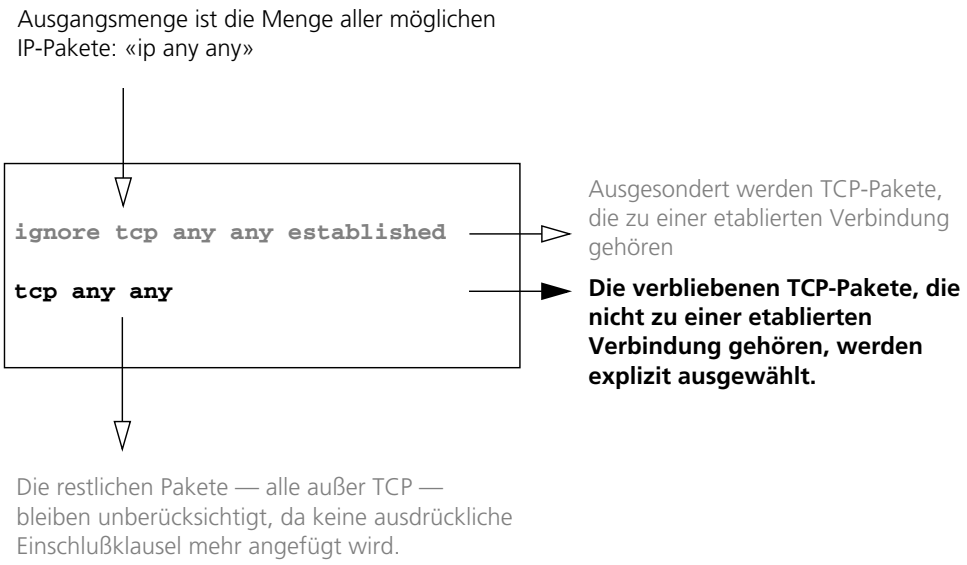

Abbildung 26 Beispiel-Restriktion: «Alle TCP-Pakete ohne 'established'-Merkmal»

any any» als unbefriedigend erweisen, so kann der Anwender die Berechnung einfach abbrechen und gezielte Analysen zu relevanten Teilmengen erstellen.

Die Spezifikation geeigneter Restriktionen wird im Prinzip genau wie beim Modul 'IngressEgress' vorgenommen, vergleiche Abschnitt «Prinzip der Filterspezifikation» (Seite 57 ff) sowie Abbildung 13 bis 15. Der Benutzer charakterisiert die gewünschte Ausgangsmenge durch eine Reihe von ACL-Klauseln, mit denen er schrittweise aus der Grundgesamtheit «ip any any» die gewünschten Teilmengen übernimmt oder aussondert, wobei der verbleibende Rest weiter zergliedert, übernommen oder verworfen werden kann.

Beispiel

Abbildung 26 zeigt ein einfaches Beispiel für eine Restriktion, die sich mittels zweier Klauseln realisieren läßt und die Teilmenge als «Menge aller TCP-Pakete, die nicht zu einer bestehenden Verbindung gehören» charakterisiert. Man erkennt, daß die Restriktionsklauseln in etwa das gleiche Format besitzen wie entsprechende Router-Konfigurationsklauseln zur Erstellung von ACLs. Allerdings wird bei der Utility 'blackwhite' das 'permit' weggelassen, und statt eines 'deny' wird das Schlüsselwort 'ignore' verwendet ('deny' ist wahlweise auch zulässig), was die Spezifikation noch etwas intuitiver macht. Abbildung 27 zeigt das Protokoll eines vollständigen Analyselaufs des Hilfswerkzeugs.

Copyright © Fraunhofer IESE 2003

```
Select an access control list:

  [1] ACL 1
  [2] ACL 102
  [3] ACL 105
  [4] ACL 106

  Which one ?: 2

Reading ACL rules '../AclRules/acl_102.xml': ... 5 rules read.

You may restrict the context for which Blacklist/Whitelist analysis.
Restrictions accumulate and may have one of these forms:

  [ignore] {any|IPADDR [WILDCAR_DMASK]}
  [ignore] PROTO SUBNET [PORTSPEC] SUBNET [PORTSPEC|ICMPSPEC] [established]

  Any context restrictions (RETURN to quit)?
  ?: ignore tcp any any established
  ?: tcp any any

No further restrictions. Processing 5 ACL rules.

  1 2 3 4 5
  5 ACL rules evaluated.

What type of result shall I print?

  [w] Whitelist
  [b] Blacklist
  [s] Shortest of these two
  [a] All: Both Whitelist and Blacklist

  Which one ?: s

Whitelist for ACL 102 will be relative to the following context
------------------------------------------------------------------

Restrictions as given:

  ignore tcp any any established
  tcp any any

Restrictions amount to:

  tcp *.*.*.*  *.*.*.* not established

Whitelist for ACL 102
------------------------------------------------------------------

tcp *.*.*.*  217.31.32.81 eq http not established
tcp 164.47.133-135.*  217.31.32.81 [0-79,81-65535] not established
```

Abbildung 27 Beispielanwendung des Hilfswerkzeugs 'blackwhite'

Copyright © Fraunhofer IESE 2003

7.2 Ergebnisaufbereitung mittels 'make_html'

Alle Analyseergebnisse werden in der zentralen Datenbank des Werkzeugs verzeichnet und am Ende des Analyselaufs im XML-Format in der Datei 'results.xml' abgespeichert. Zusätzlich wird die HTML-Darstellungskomponente aktiviert. Diese erzeugt eine übersichtliche Darstellung der Ergebnisse im Hypertext-Format (vgl. Abbildung 6, Seite 36). Mit dem Hilfswerkzeug 'make_html.pl' kann der Anwender die XML-Ergebnisdatei eines abgeschlossenen Analyselaufs ('results.xml') mit wechselnden Darstellungsparametern in HTML-Darstellung aufbereiten, ohne den Analyselauf wiederholen zu müssen.

Neue HTML-Aufbereitung nach Änderungen in 'HTML_output.conf'

Das genaue Format der HTML-Darstellung steuert die Konfigurationsdatei 'Output/HTML_output.conf'. Nach Ändern der Konfigurationsparameter ist daher eine Neuaufbereitung erforderlich. Der Aufruf des Hilfswerkzeugs ist parameterlos:

```
perl Utils/make_html.pl
```

Über die Konfigurationsdatei wird hauptsächlich die Farbzuordnung und die relative Größe der verschiedenen Schriften zueinander beeinflußt. Je nach verwendetem Browser kann die Lesbarkeit durch Ändern der Farbwahl und geschickte Fenster-Flächenaufteilung verbessert werden. Nähere Details zu den Konfigurationsmöglichkeiten finden sich in Kapitel 3.5 auf Seite 40.

Aufbereitung einer Vergleichsdarstellung von Evaluationsergebnissen

Das Hilfswerkzeug 'make_html.pl' ist auch in der Lage, die Ergebnisse zweier Analyseläufe zu vergleichen und eine Ergebnisdarstellung aufzubereiten, bei der nur solche Ergebnisse farblich hervorgehoben sind, die sich in beiden Analyseläufe unterscheiden. Eine typische Anwendung ist eine Wiederholungsprüfung nach einer Korrektur, zum Beispiel nach folgender Vorgehensweise:

1. Zunächst prüft der Anwender seine Routerkonfiguration — etwa die Datei 'routerconfiguration.cfg' — mit einem herkömmlichen Prüflauf:

   ```
   perl ./crocodile routerconfiguration.cfg
   ```

2. Er sichtet die Ergebnisse des Prüflaufs und korrigiert die Routerkonfiguration 'routerconfiguration.cfg', wo dies geboten ist. Die neue Fassung speichert er anschließend unter 'routerconfiguration-version2.cfg'.

3. Um zu sehen, wie sich die Korrekturen auf das Prüfergebnis ausgewirkt haben, sichert der Anwender die Prüfergebnisdaten des ersten Prüflaufs,

Copyright © Fraunhofer IESE 2003

zum Beispiel in das Verzeichnis '/tmp' unter dem Dateinamen 'firstrun.xml':

```
cp ./XML_data/results.xml /tmp/firstrun.xml
```

4. Der Anwender wiederholt nun den Prüflauf mit der korrigierten Routerkonfiguration 'routerconfiguration-version2.cfg':

```
perl ./crocodile routerconfiguration-version2.cfg
```

5. In der resultierenden HTML-Darstellung sieht der Anwender nun, wie CROCODILE die neue Konfiguration insgesamt bewertet. Es ist allerdings in dieser Standardansicht nur schwer erkennbar, inwieweit sich die Bewertung aufgrund der Korrekturen *verändert* hat. Um auf einen Blick eine Zusammenfassung der Änderungen zu erhalten, kann der Anwender mittels 'make_html.pl' eine Darstellung der Unterschiede relativ zum zuvor gesicherten, ursprünglichen Prüfergebnis erzeugen, indem er das Hilfswerkzeug mit dem Namen der Referenzdatei aufruft:

```
perl Utils/make_html.pl /tmp/firstrun.xml
```

Im Browser kann nun unter 'HTML_results/index.html' das Ergebnis dieser «Differentiellen Evaluation» begutachtet werden. Die Darstellung enthält alle Prüfergebnisse, aber nur solche Ergebnisse, die neu oder geändert sind, werden optisch hervorgehoben. Unveränderte Befunde werden in neutraler Färbung — üblicherweise in schwarz — wiedergegeben. Abbildung 28 zeigt ein typisches Beispiel eines Prüflaufs in Volldarstellung und in Vergleichsdarstellung. Ein Klick auf 'Source' oder 'Reference Source' im 'Evaluation Target'-Fenster (Abbildung 28 links unten) liefert bei Bedarf eine vergleichende Gegenüberstellung der beiden zugrunde liegenden Konfigurationsbeschreibungen.

Ruft der Benutzer 'make_html.pl' danach noch einmal ohne Datei-Parameter auf, so erzeugt das Hilfswerkzeug wieder die übliche Volldarstellung, ohne daß der Prüflauf wiederholt werden muß.

7.3 Generierung von URLs auf die IOS Command Reference mittels 'linkfilter'

Das Prüfwerkzeug versucht, jedem in der Konfigurationsbeschreibung erkannten IOS-Befehl passende Verweise auf Befehlsbeschreibungen in der Cisco IOS Command Reference zuzuordnen. Dazu enthält das Unterverzeichnis 'Configuration' die Dateien 'linkdata.conf' sowie 'designators.conf', die Befehlsnamen entsprechende URLs zuordnet. Das Werkzeug 'linkfilter.pl' dient dazu, aus dem aktuellen Datenbestand der Cisco IOS Command Reference 'linkdata.conf' zu generieren. Dieser Schritt ist sinnvoll, sobald neue IOS-Versionen mit geändertem Befehlsumfang zum Einsatz kommen. Die derzeitige Distribution enthält eine Datei 'linkdata.conf', die auf Basis von IOS Release 12.3 generiert wurde.

Copyright © Fraunhofer IESE 2003

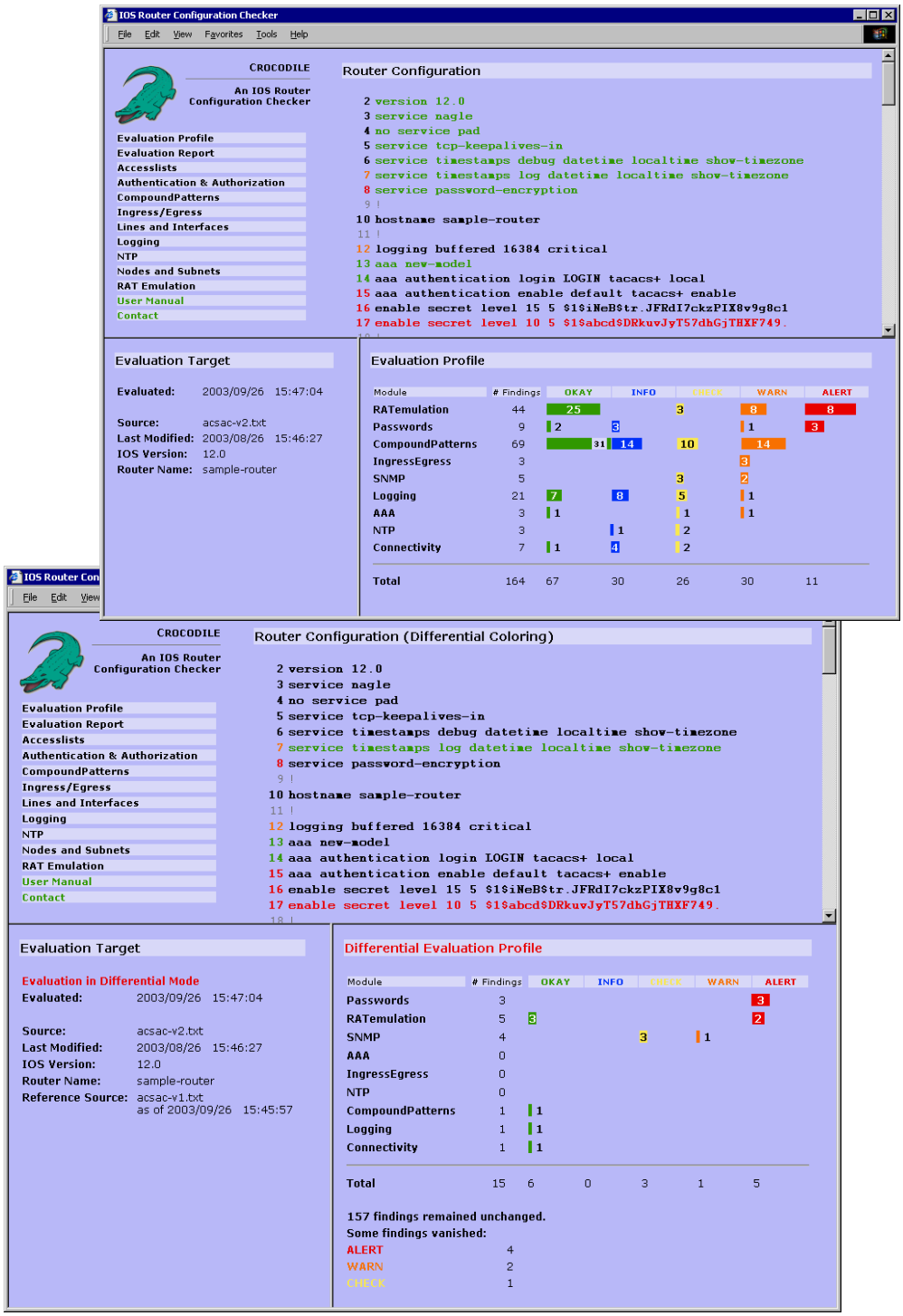

Abbildung 28 Standarddarstellung des Prüfergebnisses (oben) und vergleichende Darstellung relativ zu einem früheren Referenzergebnis (unten)

Copyright © Fraunhofer IESE 2003

Die Datenbasis des Prüfwerkzeugs umfaßt lediglich URLs, also Verweise auf die Befehlsbeschreibungen. Aus urheberrechtlichen Gründen — aber auch unter Speicherplatzgesichtspunkten — wurde darauf verzichtet, Kopien der Command-Reference-Seiten lokal bereitzuhalten. Das bedeutet, daß die im Analyselauf generierten Verweise nur dann nutzbar sind, wenn das Ergebnis auf einem Rechner mit Internet-Zugriff dargestellt wird. Ohne Internet-Anbindung können die referenzierten IOS-Seiten nicht geladen werden.

Die Aufbereitung einer neuen URL-Datenbasis 'linkdata.conf' erfordert folgende Schritte:

1. Bestimmung der URL, unter der sich der «Masterindex» der gewünschten IOS-Version befindet. (Für IOS 12.3 ist dies augenblicklich der Pfad http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123mindx/crgindx.htm)

2. Herunterladen der Masterindex-Datei unter der in 1. ermittelten URL in eine temporäre, lokale Datei (z.B. '/tmp/index.html') mit Hilfe eines Browsers.

3. Aufruf des Hilfswerkzeugs 'linkfilter.pl' mit den Argumenten <index-Datei> und <index-URL>, z.B.:

```
Utils/linkfilter.pl "/tmp/index.html" "http://www.cisco.com/…"
```

Das Hilfsprogramm erzeugt nun automatisch die Datei 'linkdata.conf' im Unterverzeichnis 'Configuration'. Es handelt sich um eine editierbare, gewöhnliche Textdatei. Jeder Eintrag belegt genau eine Textzeile in folgendem Format:

```
<ios befehl(s-prefix)>,[<kontext>],<Cisco manual name>,<URL>
```

Liegt keine Information zum Kontext eines Befehls vor, so wird im zweiten Feld "null" eingetragen.

Die automatisch generierte Datei erwies sich bisher als weitgehend vollständig und von ausreichender Qualität. Zur Optimierung wurde nur eine Handvoll zusätzlicher Einträge manuell ergänzt. Dem Anwender steht es frei, nach Belieben Zeilen hinzuzufügen oder zu löschen. Die Qualität von 'linkdata.conf' hat keinen wesentlichen Einfluß auf Analyseergebnisse des Werkzeugs, sie steigert lediglich den Bedienkomfort.

Copyright © Fraunhofer IESE 2003

8 Erstellen eigener Prüfmodule

CROCODILE ist ein erweiterbares Werkzeug. Der Anwender kann bei Bedarf eigene Prüfmodule ergänzen, um Prüfumfang und Prüftiefe zu steigern. Am Beispiel des Prüfmoduls 'Logging' skizzieren wir im folgenden die empfohlene Vorgehensweise zur Entwicklung neuer Prüfmodule.

Das Kapitel soll einen Eindruck von dem erforderlichen Entwicklungsaufwand vermitteln. Anwender des Werkzeugs, die keine Modulentwicklung betreiben wollen, können diesen Abschnitt des Handbuchs überspringen.

8.1 Recherche

Zunächst muß der Entwickler einen Prüfbereich festlegen, im Beispiel ist dies das Themengebiet «Logging». In der Regel wird man den einen oder anderen Konfigurationsbefehl kennen, der zum Prüfbereich gehört. Dieser kann als Einstieg dienen, um über die Cisco Online-Dokumentation einschlägige Beschreibungen zu ermitteln. So findet man zum Beispiel unter

```
www.cisco.com/univercd/cc/td/doc/product/software/ios122/122mindx/index.htm
```

den «Cisco IOS Release 12.2 Master Index», und dort unter dem Suchbegriff «logging command» einen Verweis auf die «Cisco IOS Configuration Fundamentals Command Reference». Ein Blick auf das Inhaltsverzeichnis zeigt schnell, daß vor allem die Seiten FR-482 bis FR-505 maßgebliche Informationen zum Prüfbereich enthalten.

Durch genaues Studium der Command Reference lassen sich unmittelbar folgende potentiell wichtige Konfigurationskommandos identifizieren:

```
logging, [default] logging buffered, logging console,
logging facility, logging history [size], logging linecard,
logging monitor, logging on, logging rate-limit,
logging source-interface, logging synchronous, logging trap
```

Daneben werden noch weitere Kommandos in Zusammenhang mit Logging erwähnt:

```
snmp-server enable traps, snmp-server host, access-list (extended)
```

Copyright © Fraunhofer IESE 2003

Inwieweit diese Kommandos tatsächlich für eine Überprüfung relevant sind, muß die nachfolgende Analyse erweisen.

8.2 Analyse

Um Prüfkriterien zu entwerfen, muß der Entwickler als nächstes die einzelnen Befehlsbeschreibungen analysieren. Selbst ein IOS-Experte hat vermutlich nicht alle Befehle und deren vielfältige Konfigurationsoptionen auswendig parat. Die Analyse stützt sich daher zunächst auf das Cisco Reference Manual, in Zweifelsfällen auf eigene Konfigurationstests an einem IOS-Router.

Ermitteln der relevanten Konfigurationsattribute

Aus den Manual-Beschreibungen ist ersichtlich, daß es fünf verschiedene Logging-Methoden gibt. Jede der Methoden hat einen fest umrissenen Satz von Konfigurationsattributen. Eine Sichtung und Klassifikation ergibt folgendes Bild bezüglich der relevanten Attribute pro Methode:

- Console Logging:
  - aktiviert/deaktiviert
  - Dringlichkeitsstufe («Level»)
- Monitor Logging:
  - aktiviert/deaktiviert
  - Dringlichkeitsstufe
- Buffer Logging:
  - aktiviert/deaktiviert
  - Dringlichkeitsstufe
  - Puffergröße
- SYSLOG Logging:
  - aktiviert/deaktiviert
  - Dringlichkeitsstufe
  - Meldungskennzeichnung («Facility»)

Copyright © Fraunhofer IESE 2003

  - Server-Adresse(n)

- SNMP Logging
  - aktiviert/deaktiviert
  - Dringlichkeitsstufe
  - History-Puffergröße

Zu jedem Attribut ist nun zu prüfen, inwieweit durch IOS bereits Voreinstellungen (Defaults) festgelegt sind. Die Attribute können dann zum Ausgangspunkt der Analyse mit entsprechenden Startwerten — gegebenenfalls mit «undefiniert» — initialisiert werden.

Festlegen der Prüfstrategie

Für jeden Konfigurationsbefehl muß der Entwickler nun anhand der Manual-Beschreibung ermitteln, welche der oben genannten Attribute von dem Befehl beeinflußt werden. Ist dieser Analyseschritt abgeschlossen, so ist damit die Prüfmodul-Logik in ihren Grundzügen umrissen:

- Für jedes Logging-Attribut wird im Prüfmodul eine Variable angelegt, die den jeweiligen Attributstatus widerspiegelt; der Initialwert wird entsprechend dem Default gesetzt.

- Für jeden Konfigurationsbefehl definiert der Entwickler ein passendes Pattern und ordnet ihm einen Pattern Handler zu. Der Handler wertet die Befehlsparameter aus und führt die Belegungen der betroffenen Attributvariablen geeignet nach.

- Sind alle Konfigurationsbefehle ausgewertet, so ermittelt das Prüfmodul den resultierenden Logging-Status anhand der finalen Attributbelegungen und erzeugt entsprechende Prüfprotokolleinträge.

Die Durchführung dieser Punkte ist im Prinzip einfach, in der Praxis ergeben sich dabei jedoch einige Schwierigkeiten. Dies hängt vor allem damit zusammen, daß die Cisco-Dokumentation oftmals mehrdeutig ist. Nicht immer ist klar, welche Attribute genau von einer Konfigurationsoption betroffen sind: Führt das Weglassen eines optionalen Parameters zum Rücksetzen des betroffenen Attributs auf den Default-Wert? Ist der Default für verschiedene Router-Modelle unterschiedlich? Sind alle syntaktisch erlaubten Parameter-Kombinationen eines Befehls in der Praxis auch zulässig? — Solche und ähnliche Unsicherheiten erschweren die Analyse. Eine Klärung ist oft nur durch Ausprobieren in einer IOS-Sitzung an einem Router möglich.

Copyright © Fraunhofer IESE 2003

8.3 Implementieren der Prüflogik

Aufgrund der Recherche ist ungefähr klar geworden, welche Prüfschritte erforderlich sind und wie die Vorgehensweise dabei ist. Nun gilt es, die Prüfstrategie in PERL-Code umzusetzen.

Prüfmodulgerüst

Ausgangspunkt jeder Neuentwicklung ist ein Codegerüst, das der Entwickler durch «Cut and Paste» aus einem vorhandenen Modul leicht erzeugen kann. Abbildung 29 zeigt eine passende Vorlage.

Das abgebildete Codefragment ist im Prinzip ein vollwertiges Prüfmodul; es ist unter CROCODILE lauffähig, sobald in der ersten Zeile der Package-Name eingetragen wird. Alle wesentlichen Moduleigenschaften, die für die Kompatibilität mit dem Werkzeug erforderlich sind, erbt das Modul von der Basisklasse 'Plugin::Module'. Allerdings sind noch keine Patterns eingetragen: Die generische Handler-Routine käme daher nicht zur Ausführung; lediglich die (leere) Nachverarbeitungsroutine 'summary()' wird am Ende des Analyselaufs vom CROCODILE-Framework aktiviert.

Der Entwickler muß nun das Codegerüst schrittweise gemäß der in Kapitel 8.2 skizzierten Prüfstrategie mit Prüflogik füllen.

Repräsentation des Router-Konfigurationszustands

Für die in Kapitel 8.2 ermittelten Konfigurationsattribute wird jeweils eine lokale Zustandsvariable eingeführt. Diese wird mit dem vorgesehenen Default-Wert initialisiert, zum Beispiel:

```
## ---------- Local state variables for bookkeeping -----------

# Console/terminal logging

my $console_log = 1;            # Logging to console?
my $console_level = 7;
my $monitor_log = 1;            # Logging to monitor lines?
my $monitor_level = 7;

# SNMP logging

my $history_level = 4;          # Minimum level sent to SNMP as trap
my $history_size = 1;           # History buffer size
my $traps_enabled = 0;          # Are SNMP traps sent at all?
…
```


Copyright © Fraunhofer IESE 2003

```
package Plugin::<Module Name>;
use strict;

use Plugin::Module;
use vars qw(@ISA);
@ISA = ("Plugin::Module");

use Auxiliary::Logging qw(:user);
my $macroref = [];

my $patternref = { };

## ---------- Local state variables for bookkeeping -----------

## ---------- Public Functions -------------------------------

sub new {
  my ($proto,@data) = @_;
  my $class = ref($proto) || $proto;

  # Idempotency: At most one instance is created of this class
  my $self=$class->SUPER::does_exist();
  if (!defined $self) {
    $self = $class->SUPER::new(@data);
    $self->{MACROS} = $macroref;
    $self->{PATTERNS} = $patternref;
    $self->{POSTPROCEDURE} = \&summary;
    }

  return $self;
}

## ---------- Module-specific handlers ------------------------

sub generic_handler {
  my ($self, $match) = @_;

  $self->_dump_hashref($match, "_generic_handler");
  $self->add_logical_context($match->{LINE_NO},"<Some Context>");
  $self->attach_ciscolinks($match->{LINE_NO}, $match->{MATCH});
}

## ---------- Postprocessing Specifications -------------------

sub summary {
  my ($self) = @_;
}
```

Abbildung 29 Vorlage zur Erstellung eines neuen Prüfmoduls

Copyright © Fraunhofer IESE 2003

Auswahl und Test der Patterns

Als nächstes muß der Entwickler für jeden relevanten Befehl ein passendes Pattern eintragen. Zur Orientierung dient das Cisco Reference Manual. Allerdings lassen sich die dort beschriebenen Formate in der Regel nicht wörtlich übertragen. Sie bedürfen meist einer Anpassung, etwa aus folgenden Gründen:

- Oft wird angestrebt, mit einem einzigen Pattern sowohl die positive, als auch die negierte Form eines Patterns zu erfassen, also zum Beispiel:
  logging rate-limit {*number*|**all**|**console**} [**except** *severity*]
  no logging rate-limit.
  Dazu ist es erforderlich, die Befehlsparameter in einer zusätzlichen []-Klammerung einzuschließen, um sie als optional zu kennzeichnen:
  logging rate-limit [{*number*|**all**|**console**} [**except** *severity*]]
  Die Negation selbst — das Schlüsselwort 'no' — wird von CROCODILE automatisch als optional angesehen und in jedem Falle toleriert.

- Die im Manual ausgewiesene Syntax ist oftmals irreführend, wenn nicht gar falsch! So lautet zum Beispiel die korrekte Darstellung der Syntax für 'logging rate-limit' wie folgt (zumindest für einen von uns überprüften IOS-Router):
  logging rate-limit [**console** [**all**]|**all**] *number* [**except** *severity*]
  Man vergleiche dies mit den Angaben im Cisco Manual auf Seite FR-499.

- Mitunter wird ein Pattern absichtlich allgemeiner ausgelegt, als es die korrekte Befehlssyntax eigentlich erfordert. Dies kann dazu dienen, das Pattern zu vereinfachen, ist aber auch hilfreich, um Tippfehler oder Bereichsverletzungen im Konfigurationstext aufzuspüren. So erkennt CROCODILE mittels
  logging history size NUM
  zum Beispiel auch falsche Konfigurationsbefehle mit ungültiger Größenangabe, während
  logging history size 1-500
  zwar genauer ist, aber bei Überschreitung der erlaubten Maximalgröße 500 nicht anspringt. Damit der Pattern Handler auch bei solchen Fehlern getriggert wird, lohnen sich oft Verallgemeinerungen.

Der Entwickler kann seine Patterns in das Codefragment eintragen und deren Funktionsfähigkeit prüfen. Zur Formulierung von Patterns kann der Entwickler geeignete Makros (vgl. Kapitel 3.2, Seite 26) einführen. Für einen Testlauf wird jedem Pattern zunächst der generische Handler 'generic_handler()' zugeordnet:

```
my $macroref = [
    INTERFACE => "IF_TYPE IF_NUM",
    …
    ];
```


Copyright © Fraunhofer IESE 2003

```
my $patternref = {
     "logging source-interface [ INTERFACE ]" => {
       HANDLER => [ \&generic_handler ],
       },
     "logging history size NUM" => {
       HANDLER => [ \&generic_handler ],
       },
     …
     };
```

Sind alle Patterns korrekt eingetragen, dann kann der Entwickler das neue Prüfmodul mittels 'plugin()' ins CROCODILE-Hauptprogramm einklinken. Anhand von Beispielkonfigurationen sind jetzt erste Tests möglich:

- Werden alle Konfigurationsbefehle (in allen möglichen Varianten) tatsächlich erkannt? Auch in negierter Form?

- Ist eines der Patterns versehentlich mehrdeutig und führt zu Konflikten, die durch Pattern-Modifikation ausgeräumt werden müssen? (Vgl. Kapitel 3.2, Abschnitt «Konflikte: Gierige oder minmale Pattern-Interpretation?» auf Seite 24)

- Wird jedem erkannten Befehl mittels 'attach_ciscolinks()' auch eine sinnvolle Manual-Referenz zugeordnet? (Gegebenenfalls muß der Suchbegriff angepaßt oder die Linkdatenbank im Unterverzeichnis 'Configuration' ergänzt werden, vgl. dazu Kapitel 7.3.)

Erst wenn alle Patterns zufriedenstellend funktionieren, sollte mit der Erstellung der Prüflogik begonnen werden.

Erstellen der Handler-Prüflogik

Pattern für Pattern tauscht der Entwickler nun den generischen Handler 'generic_handler()' gegen eine maßgeschneiderte Handler-Routine aus. Der generische Handler-Code dient dabei als Vorlage.

Über den Parameter '$match' hat der Handler Zugriff auf den Konfigurationsbefehl, der durch seine Übereinstimmung mit dem Pattern den Handler getriggert hat. Die Prüflogik beginnt in der Regel damit, den «Match» mittels 'add_logical_context()' bzw. 'set_clause_context()' einem Analysekontext zuzuweisen (der vorangehende Aufruf 'dump_hashref()' in der Vorlage gemäß Abbildung 29 protokolliert den '$match' Datensatz auf die Konsole — vergleiche Abbildung 5 auf Seite 31 — und dient nur zur Orientierung während der Entwicklung; im fertigen Prüfmodul wird dieser Befehl auskommentiert oder gelöscht). Außerdem wird versucht, mittels 'attach_ciscolinks()' eine URL zur Befehlsbeschreibung zuzuordnen.

Copyright © Fraunhofer IESE 2003

```
sub logging_source {
  # logging source-interface [ INTERFACE ]
  my ($self, $match) = @_;

  #-   $self->_dump_hashref($match, "logging_source");

  $self->add_logical_context($match->{LINE_NO},"Logging");
  $self->add_logical_context($match->{LINE_NO},"SYSLOG");
  $self->attach_ciscolinks($match->{LINE_NO},
                           "logging source-interface");

  if (defined $match->{NEGATION}) {
    $source_interface = undef;
    if (defined $match->{"2"}) {
        $self->add_annotation($match->{LINE_NO},
                              "no params expected",
                              'WARN');
        }
    }
  elsif (defined $match->{"2"}) {
    $source_interface = $match->{"2"};
    $self->change_criticality($match->{LINE_NO}, 'OKAY');
    }
  else {
    $source_interface = undef;
    $self->add_annotation($match->{LINE_NO},
                          "parameter expected",
                          'ALERT');
    }
}
```

Abbildung 30 Beispiel für eine Handler zum Pattern '**logging source-interface** [INTERFACE]'

Als nächstes wird in der Regel eine Fallunterscheidung getroffen, ob der Konfigurationsbefehl in negierter oder nicht-negierter Form vorliegt. In beiden Fällen erfolgt danach üblicherweise eine Korrektheits- und Plausibilitätsprüfung der Befehlsparameter. Ungereimtheiten werden in der Ergebnisdatenbank notiert. Dazu stehen verschiedene Ausgabemethoden zur Verfügung, die das Prüfmodul von seiner Basisklasse 'Plugin::Module' erbt (siehe Anhang A).

Sind alle Prüfungen abgeschlossen und der Konfigurationsbefehl erweist sich als zulässig, so aktualisiert der Handler die Zustandsvariablen des Prüfmoduls entsprechend der Wirkungsweise des Konfigurationsbefehls. Abbildung 30 zeigt einen typischen Handler des 'Logging'-Prüfmoduls. Je nach Befehlsparameter können die notwendigen Korrektheits- und Plausibilitätsprüfungen wesentlich komplexer ausfallen als in diesem Beispiel.

Copyright © Fraunhofer IESE 2003

```
sub summary {
  my ($self) = @_;

  if (! defined $linecard_log ) {
    $self->add_context_annotation("Logging",
                "Unclear whether linecard logging is enabled",
                'WARN');
    }
  elsif ($linecard_log == 0) {
    $self->add_context_annotation("Logging",
                "Linecard logging is disabled",
                'INFO');
    }

  …

  # Create Bookmark label 'Logging' and assign context refs

  $self->add_bookmark("Logging","Logging (General)",
                      "Logging", "toContext");
  $self->add_bookmark("Logging", "Log Buffer",
                      "Log Buffer", "toContext");
  $self->add_bookmark("Logging", "SYSLOG",
                      "SYSLOG", "toContext");
  $self->add_bookmark("Logging", "SNMP Logging",
                      "SNMP Logging", "toContext");
}
```

Abbildung 31 Ausschnitt aus der "Logging'-Nachverarbeitungsroutine

Erstellen der Prüflogik für die Befund-Zusammenfassung

Erst nachdem alle Zeilen der Routerkonfiguration analysiert worden sind steht fest, in welchen Zustand die Konfiguration den Router versetzt. Daher kann ein einzelner Handler nur vorläufige Erkenntnisse zum Befund beitragen. Für eine abschließende Würdigung wird eine gesondere Routine aufgerufen. Ihr Name ist im Prüfmodul unter '$self->{POSTPROCEDURE}' hinterlegt. In unserem Beispiel übernimmt 'summary()' diese Aufgabe.

Die Nachverarbeitung analysiert die Endwerte der Zustandsvariablen und zieht daraus Rückschlüsse auf den Konfigurationszustand des Routers. Die sich ergebenden positiven oder negativen Befunde werden in die Befunddatenbank eingetragen. Dazu stehen dem Entwickler die Methoden laut Anhang A zur Verfügung. Abbildung 31 zeigt einen typischen Ausschnitt der Nachverarbeitungsroutine (für eine ausführliche Darstellung verweisen wir auf den Quellcode des 'Logging' Prüfmoduls).

Copyright © Fraunhofer IESE 2003

Die Nachverarbeitungslogik komplettiert das Prüfmodul. CROCODILE verfügt nun über einen neuen Prüfbereich. Der Entwickler sollte das neue Modul vor der Freigabe ausgiebig an Testfällen und realen Praxisbeispielen testen.

Protokollierung der Abläufe

Das Prüfmodul erfaßt und protokolliert seine Befunde mit den Methoden der Datenbankschnittstelle (siehe dazu Anhang A). Daneben sind manchmal aber auch Rückmeldungen zum Ablauf der Berechnung erforderlich. Solche Meldungen werden in jedem Falle in die Logdatei eingetragen, soweit eine solche deklariert ist; je nach eingestellter «Verbosity» (Option '-v') erscheinen die Meldungen auch im Konsolenfenster.

Damit sich die Konsolenausgabe per Option einschränken und in der Protokolldatei erfassen läßt, muß der Entwickler bei der Programmierung spezielle Ausgabeoperationen als Ersatz für das Perl-typische 'print' benutzen. Diese Operationen stellt die 'user'-Schnittstelle des Hilfsmoduls 'Auxiliary/Logging.pm' bereit, welche Prüfmodule importieren müssen, um Zugriff auf die Methoden zu erhalten:

```
package Plugin::<Module Name>;
use strict;
use Auxiliary::Logging qw(:user);
…
```

Für die Protokollierung des Rechenfortschritts und die Ausgabe von Warn- oder Fehlermeldungen sind folgende Methoden verfügbar:

- console <*List*>
  <*List*> wird auf die Konsole ausgegeben, nicht jedoch in der Logdatei erfaßt (entspricht etwa 'print STDERR <*List*>, "\n"').

- trace <*List*>
  <*List*> wird — unabhängig von der eingestellten «Verbosity» — sowohl auf die Konsole als auch in die Logdatei geschrieben.

- info <*List*>
  <*List*> wird in die Logdatei geschrieben und bei «Verbosity»-Level 1 oder 2 zusätzlich auf der Konsole ausgegeben.

- verbose <*List*>
  <*List*> wird in die Logdatei geschrieben und bei «Verbosity»-Level 2 zusätzlich auf der Konsole ausgegeben.

Copyright © Fraunhofer IESE 2003

- warning *<List>*
  <List> wird als Warnung (d.h. mit vorangestelltem '*** ' in jeder Ausgabezeile) in die Logdatei und auf die Konsole geschrieben.

- error *<List>*
  <List> wird als Error (d.h. mit vorangestelltem '*** ERROR: ' in jeder Ausgabezeile) in die Logdatei und auf die Konsole geschrieben.

- abort *<List>*
  <List> wird als Warnung (d.h. mit vorangestelltem '*** ' in jeder Ausgabezeile) in die Logdatei und auf die Konsole geschrieben. Danach wird die Programmausführung[16] mit einer Meldung und entsprechendem Exit-Status abgebrochen.

Die Ausgabemethoden sind so realisiert, daß sie syntaktisch genau wie ein gewöhnliches 'print' (allerdings ohne Dateideskriptor-Argument!) verwendet werden können. Das heißt zum Beispiel, daß die Argumentliste *<List>* nicht in Klammern eingeschlossen werden muß und daß beliebig viele Ausgabe-Elemente — durch Komma getrennt — angegeben werden dürfen.

Anders als 'print' fügen die oben genannten Methoden am Ende der Ausgabe automatisch einen Zeilenvorschub "\n" an. Sollte dies in Ausnahmefällen einmal unerwünscht sein, so kann der Zeilenvorschub unterdrückt werden, indem man das erste Listenelement mit einem '^' einleitet, etwa:

```
warning '^This is part 1 of a multi-part output line; ';
warning 'this is part 2.';
warning 'And here is part 3.';
```

Diese drei Kommandos erzeugen folgende Ausgabe:

```
*** This is part 1 of a multi-part output line; this is part 2.
*** And here is part 3.
```

Die mit 'warning' oder 'error' ausgegebenen Meldungen werden automatisch gezählt. Das Zählergebnis wird in der Abschlußmeldung des Programmlaufs ausgewiesen. Mittels 'get_counts' lassen sich die Zählerstände im Programm jederzeit abfragen:

```
my ($warn_count, $error_count) = get_counts();
```

16 Im Batch-Modus bewirkt ein 'abort' in einem Prüfmodul und seinen Hilfsmodulen nur den Abbruch eines einzelnen Jobs; das umgebende Stapelverarbeitungs-Frontend «fängt» den Fehler und setzt die Verarbeitung mit dem nächsten anstehenden Job fort.

Copyright © Fraunhofer IESE 2003

Um den Rechenfortschritt anzuzeigen, können mittels 'Logging.pm' zusätzlich «überschreibbare» Meldungen ausgegeben werden. Der Meldungstext erscheint nur auf der Konsole und wird vom nächsten Ausgabekommando getilgt und überschrieben. Zwei Operationen sind verfügbar:

- print_eraseable <*List*>
  <*List*> wird auf die Konsole geschrieben; der Cursor wird ohne Zeilenvorschub auf den Textanfang zurückgesetzt. (<*List*> darf keine Zeilenvorschübe "\n" enthalten!)

- print_progresswheel
  Es wird ein «Laufrad» in Form einer sich auf der Stelle drehenden «Speiche» auf der Konsole ausgegeben; jeder Aufruf dreht die Speiche eine Stellung weiter: '|', '/', '–', '\', …

Überschreibbarer Text — ob Meldung oder Laufrad — wird von nachfolgenden Meldungen überschrieben, sowohl von normalem wie überschreibbarem Text. Damit das Tilgen und Zählen von Meldungen ordnungsgemäß funktioniert, müssen allerding durchgängig die Methoden von 'Logging.pm' verwendet werden. Befehle des Typs 'print [STDOUT | STDERR]' sollten im fertigen Programmcode nicht erscheinen. In der Debug-Phase der Programierung können sie jedoch ohne weiteres verwendet werden.

8.4 Feinschliff

Die Vorgehensweise gemäß den Kapiteln 8.1 bis 8.3 liefert in der Regel noch kein perfektes Prüfmodul. In der praktischen Erprobung ergeben sich oft noch Modifikationen und Ergänzungen, zum Beispiel:

- Die Zuordnung von Kritikalitätsstufen (ALERT, WARN, CHECK usw.) bedarf oft noch der Feinjustierung, um eine konsistente Befundeinstufung aller Prüfmodule zu erzielen.

- Oft ergeben sich Möglichkeiten für zusätzliche Prüfungen, indem man einzelne Patterns nachträglich verallgemeinert und dadurch zusätzliche Fehlerfälle erfaßt.

- Die ursprünglich ins Auge gefaßte Menge von relevanten Konfigurationsbefehlen erweist sich nachträglich mitunter als unvollständig. So zeigt sich bei näherer Betrachtung zum Beispiel, daß dem Themenbereich 'Logging' unter anderem auch noch folgende Befehle zuzuordnen sind:
  `mpls traffic-eng logging tunnel, mpls traffic-eng logging lsp`
  Dafür sind zusätzliche Zustandsvariablen, Patterns und Handler zu ergänzen.

Copyright © Fraunhofer IESE 2003

- Zusätzlich kann das Prüfmodul auch analysieren, ob sich einzelne Konfigurationsklauseln gegenseitig aufheben oder ob eine unzulässige Reihenfolge von Konfigurationsbefehlen vorliegt. Dies erfordert diffizilere Prüflogik, die mit den naheliegenden Zustandsvariablen allein nicht realisiert werden kann.

- Der Prüfbefund läßt sich gezielt mit (Quer-)Verweisen anreichern. Dazu können zusätzliche logische Kontexte, Bookmarks, Dump Areas oder gar Korrekturvorschläge (Fix Suggestions) generiert werden. Verweise auf externe Dokumentation — etwa in Form einer URL auf ein White Paper im Internet — sind ebenso vorstellbar.

Der Entwickler muß entscheiden, wo hier das rechte Verhältnis zwischen Aufwand und Nutzen liegt. Die Entscheidung hängt nicht zuletzt davon ab, welche Fachkenntnis der Entwickler dem vorgesehenen Nutzerkreis unterstellt: Je fachkundiger der Anwender, um so spärlicher kann die Befundaufbereitung ausfallen.

Ein Blick auf den Originalcode des Prüfmoduls zeigt, wieviele Ergänzungen schließlich noch stattgefunden haben, um das Modul abzurunden.

8.5 Anmerkungen

Das Prüfmodule 'Logging' gehört zu den einfacheren, wenn auch länglichen Modulen des CROCODILE-Rahmenwerks. Seine Prüflogik ist nur von mäßiger Komplexität. Insofern beschreibt dieses Kapitel ein einfaches Beispiel, zudem ist die Darstellung auf das Wesentliche verkürzt. Tatsächlich erforderten Konzeption, Bereitstellung und Test des Prüfmoduls 'Logging' einige wenige Personentage.

Je nach Prüfbereich kann die Entwicklung eines Prüfmoduls wesentlich aufwendiger sein. Maßgebliche Einflußfaktoren für den Entwicklungsaufwand sind vor allem:

- Wie viele Konfigurationsbefehle umfaßt der Prüfbereich? Wie umfangreich können diese Befehle parametrisiert werden?

- Wie unabhängig sind die relevanten Konfigurationsbefehle von einander? Inwieweit sind Wechselwirkungen zu beachten (z. B. eine vorgeschriebene Reihenfolge von Konfigurationsschritten)?

- Wie eindeutig und vollständig ist die von Cisco bereitgestellte Dokumentation? (Bzw.: Wie gut ist der Entwickler mit dem Prüfbereich vertraut?)

Copyright © Fraunhofer IESE 2003

Gerade der letztgenannte Punkt ist häufig der entscheidende «Kostentreiber» für die Bereitstellung eines neuen Prüfmoduls, denn die Cisco-Dokumentation läßt allzu oft wichtige Fragen unbeantwortet. Zuweilen führen die Unterlagen sogar regelrecht in die Irre! Es bedarf dann umfangreicher Routertests, um sich Klarheit über die tatsächliche Wirkungsweise einzelner Konfigurationsbefehle zu verschaffen.

Sofern der Entwickler den Prüfbereich genau versteht, sollte er — mit ein wenig Routine — nach der hier dargestellten Vorgehensweise sehr zügig und systematisch ein passendes Prüfmodul bereitstellen können.

Copyright © Fraunhofer IESE 2003

9 Status und Ausblick

9.1 Status der Werkzeugunterstützung

Das CROCODILE-Framework hat sich in bisherigen Anwendungen recht gut bewährt, obwohl es längst noch nicht in allen Bereichen vollständig ausgebaut ist. In den meisten Beispielkonfigurationen aus der Praxis, die wir mit CROCODILE analysiert haben, fanden sich zumindest Ungereimtheiten — oft sogar gravierende Fehler! Wenn unsere Beobachtung repräsentativ ist, so sind Konfigurationsmängel eher die Regel als die Ausnahme, und Sicherheitslücken bleiben oft über Monate unentdeckt. Die Tatsache, daß CROCODILE Schwachstellen aufdeckt, die selbst geschultes Personal übersehen hat, bestärkt uns in unserem Ansatz.

Die derzeit erzielte Abdeckung von Sicherheitsanforderungen durch Prüfkriterien fußt maßgeblich auf dem generischen Prüfmodul 'CompoundPatterns'. Es hat sich gezeigt, daß Prüfanforderungen unterschiedlichster Art leicht im Formalismus von 'CompoundPatterns' spezifizierbar sind. Der Ansatz, eine einfache Metasprache zur Deklaration von Prüfanforderungen einzuführen, hat sich als sehr hilfreich erwiesen.

'CompoundPatterns' betont die Einfachheit der Prüfpunktdeklarationen, was zu Lasten der Analysemächtigkeit und Protokollierungsgenauigkeit dieses Prüfmoduls geht. Eine noch komplexere Spezifikationssprache zur Beschreibung von Prüfpunkten wäre vielseitiger, würde jedoch tiefere Kenntnisse und größeren Aufwand für den Revisor mit sich bringen. Die Praxis muß erweisen, was der ausgewogenste Kompromiß zwischen Mächtigkeit und Bedienfreundlichkeit eines generischen Moduls ist. Das Modul 'IngressEgress' demonstriert, wie sich flexible, zugleich mächtige Prüfanforderungen mit einfacher Bedienbarkeit vereinbaren lassen — wenn man die Prüfproblem-Klasse geeignet einschränkt.

Das Ausgabeformat zur Darstellung der Prüfergebnisse ist in unseren Augen recht ausgereift. Der Rückgriff auf HTML-Hypertext ermöglicht flexible Anpassungen der Formatgestaltung an die Bedürfnisse des Anwenders; die Verwendbarkeit eines beliebigen Standard-Browsers sorgt für sehr gute Portierbarkeit auf unterschiedlichste Plattformen.

Im Stapelverarbeitungsmoduls liefert CROCODILE seine Ergebnisse wahlweise auch im CSV-Format, was eine einfache Weiterverarbeitung mit unabhängigen Werkzeugen gewährleistet.

Copyright © Fraunhofer IESE 2003

9.2 Nächste Schritte

Die vorhandenen Prüfmodule decken nur einige — wenn auch wichtige — Teilbereiche der Routersicherheit ab. Durch zusätzliche Module lassen sich bestehende Lücken in der Prüflogik schließen, die mit den einfachen Mitteln von 'CompoundPatterns' nur unvollständig behandelbar sind. Zudem ist es wünschenswert, zu einzelnen Mängeln sehr spezifische Diagnosemeldungen zu generieren, was 'CompoundPatterns' nicht ermöglicht.

Einige Prüfbereiche erfordern vermutlich parametrisierbare Prüfmodule, die mit einer Modulkonfiguration in geeignetem Beschreibungsformat bestückt werden müssen. Zukünftige Untersuchungen müssen erweisen, wie der Anwender Prüfkriterien zu bestimmten Prüfbereichen am sinnvollsten spezifizieren kann.

Für die planmäßige Erweiterung des Prüfumfangs bietet das CROCODILE-Framework hervorragende Voraussetzungen. Allerdings ist der erforderliche Realisierungsaufwand nur durch entsprechende Nachfrage zu rechtfertigen: Prüfbereiche nur «auf Verdacht» zu erschließen, ohne daß absehbar ein Kunde danach verlangt, ist nur in wenigen Kernbereichen vertretbar.

Andererseits kommt jedes neue Prüfmodul allen CROCODILE-Anwendern zugute, denn es kann der gesamten Kundschaft verfügbar gemacht werden. Zum Vorteil aller Anwender hoffen wir daher auf einen ausreichend großen Kundenstamm, der eine kostengünstige Weiterentwicklung des Frameworks sichert.

9.3 Ausdehnung auf andere Netzwerkkomponenten

CROCODILE ist hinreichend flexibel, um auch auf andere Konfigurationsformate angewendet zu werden. Naheliegend wären Cisco PIX Firewall-Konfigurationen, die im Vergleich zu IOS-Konfigurationen eine etwas andere, jedoch recht ähnliche Befehlslogik verwenden. Dies würde zwar eine völlige Neugestaltung der Pattern und Pattern Handler erfordern; die Grundzüge der Werkzeugumgebung, insbesondere Parser und HTML-Ausgabe, blieben jedoch unverändert erhalten und wären nutzbringend wiederverwendbar.

Die Bereitstellung von PIX-Prüfmodulen ist als ein Weiterentwicklungsschritt fest ins Auge gefaßt. Grundsätzlich sollten auch Router oder Firewalls anderer Hersteller als Cisco analysierbar sein, soweit deren Konfiguration in Form von textuellen Beschreibungen mit nachvollziehbaren Syntaxregeln vorliegt. Konkrete Pläne dazu gibt es derzeit jedoch nicht.

Copyright © Fraunhofer IESE 2003

A Schnittstelle zur Befunddatenbank

Die folgende Beschreibung der Schnittstelle Prüfmodul–Befunddatenbank richtet sich vornehmlich an Benutzer, die bestehende Prüfmodule modifizieren oder neue Prüfmodule realisieren wollen.

Alle Prüfmodule speichern ihre Befunde in der zentralen Befunddatenbank ab. Die Datenbank wird durch das PERL-Modul 'Auxiliary/ResultData.pm' realisiert. Sie verfügt über Methoden für das Speichern der unterschiedlichen Ergebnisdatentypen: Anmerkungen, Befundkategorien, Verbesserungsvorschläge usw. — Erläuterungen dazu finden sich in Kapitel 3.4 auf Seite 32.

Datenbankschnittstelle der Prüfmodule

Alle Zugriffsmethoden der Befunddatenbank sind Schreiboperationen. Es ist nicht vorgesehen, Anfragen an die Befunddatenbank zu stellen. Das vereinfacht die Schnittstelle und erleichtert den Modulen die Interaktion mit der Datenbank.

Prüfmodule nutzen die Datenbankschnittstelle in der Regel nicht direkt, sondern mittels Wrapper-Methoden, die von der gemeinsamen Basisklasse aller Prüfmodule — Plugin::Module — mit verkürzter Parameterliste bereitgestellt werden. Die Methoden im einzelnen sind:

- set_clause_context($linenumber,$context_name)
  Eine vom Parser erkannte Zeile kann hiermit einem Klauselkontext zugeordnet werden. Jede Zeile kann — bedingt durch die IOS Konfigurationsmodi — nur einem Klauselkontext angehören! Als Name für den Kontext empfiehlt es sich, sich an die Konvention zu halten, die Zeilennummer der den Kontext einleitenden Konfigurationsklausel zu benutzen (also z.B. die Zeile, in der das Schüsselwort 'Interface' steht, für alle Interface-Konfigurationsklauseln der jeweiligen Schnittstelle). Ein Kontext muß nicht explizit erzeugt werden, sondern er entsteht automatisch, sobald ihm eine Zeile zugeordnet wurde.

- change_criticality($linenumber,$severity [,$force_flag])
  Hiermit kann die Kritikalität der vom Parser erkannten Zeile geändert werden. Allerdings wird diese Methode in aller Regel implizit durch den Aufruf von 'add_annotation' ausgeführt, so daß eine explizite Nutzung

Copyright © Fraunhofer IESE 2003

dieser Methode eher die Ausnahme ist. Trotzdem ist es wichtig, den Umgang mit Kritikalitäten zu verstehen.
Jedes Modul gibt jeweils seine individuelle Kritikalitätseinstufung zu einer Konfigurationsklausel an. Erst das Ausgabemodul ermittelt daraus, wie die Gesamtkritikalität unter Berücksichtigung aller Modul-Einstufungen für diese Zeile lautet. Allerdings muß sich jedes Modul für eine Einstufung pro Zeile entscheiden. Wird die Funktion von einem Modul mehrfach mit unterschiedlicher Kritikalität pro Zeile aufgerufen, so wird nicht der alte Wert durch den Neuen überschrieben, sondern es setzt sich immer der schlechteste durch (also ALERT vor WARN usw.). Es kann aber nun vorkommen, daß gerade das «blinde» Überschreiben des alten Wertes gewünscht ist, so daß die neue Kritikalität auf jeden Fall übernommen wird. In diesem Fall kann man das 'force'-Flag setzen (d.h. das letzte Argument mit irgendeinem Wert füllen). Hierbei ist zu beachten, daß die somit angegebene Kritikalität zwar vorerst uneingeschränkt übernommen wird, falls aber im Verlauf der Analyse ein weiteres Mal ein 'change_criticality()' ohne force-Flag aufgerufen wird, wird wieder der schlechteste Wert der beiden Kritikalitäten übernommen. Für 'severity' sind die Werte 'OKAY', 'INFO', 'CHECK', 'WARN' oder 'ALERT' zulässig (dies ist die Reihenfolge vom besten zum schlechtesten Wert).

- set_category($linenumber,$category)
  Mit dieser Methode kann man jeder Zeile einer Kategorie zuordnen. Welche Begriffe hier als Kategoriename ('class') verwendet werden, ist dem Programmierer freigestellt. Ist die Kategorie einmal von einem Modul festgelegt worden, so kann dieses sie nicht mehr ändern.

- add_annotation($linenumber,$text,$severity)
  Mit dieser Methode können Anmerkungen zu einer Zeile hinterlegt werden. Jeder Aufruf dieser Methode fügt in der Befunddatenbank einen neuen Anmerkungssatz zu dieser Zeile hinzu. Der Anmerkung wird eine angegebene Kritikalität zugeordnet. Der Aufruf von 'add_annotation()' impliziert 'change_criticality()', so daß sich mit dem Hinzufügen einer Anmerkung auch die Kritikalität der Zeile ändert.

- add_reference($linenumber,$ref_name,$reference,$ref_type)
  Der Aufruf ordnet einer Konfigurationsklausel einen Verweis zu. Jede Zeile kann beliebig viele Verweise haben. Als Argumente werden hier nach der Zeilennummer, an die der Verweis angehängt werden sollen, noch der sichtbare Verweisname, der Verweis (also z.B. eine URL oder ein Dateiname) und der Typ des Verweises angegeben. Mögliche Typen sind 'file', 'URL', 'DumpData', 'toLine' und 'toContext'. Der erste Typ öffnet eine Datei, beim zweiten wird auf einen URL verlinkt. Der dritte zeigt auf einen Text vom Typ DumpData (als '$reference' ist genau der Name des Dumpdata-Objektes anzugeben). Beim Typ "toLine", kann auf eine beliebige Zeile der Konfigu-

Copyright © Fraunhofer IESE 2003

rationsbeschreibung verwiesen werden, und beim Typ "toContext" wird der entsprechende Kontext angezeigt (als '$reference' muß der Kontextname angegeben werden), wobei es sich sowohl um einen logischen Kontext als auch um einen Klauselkontext handeln kann. Bei den letzten drei Typen ist es außerdem wichtig, daß die entsprechende Datenstruktur bereits angelegt ist, bevor der Verweis gesetzt wird. Es kann also z.B. nicht auf eine Zeile verwiesen werden, die vom Parser (noch) nicht erkannt wurde. Auch Kontexte bzw. Dumps müssen vorher bereits erstellt worden sein.

- add_logical_context($linenumber,$context_name)
  Hiermit kann jede Zeile einem beliebigen logischen Kontext zugeordnet werden, um Zusammenhänge zwischen Konfigurationsklauseln zu verdeutlichen. Der Name des Kontexts unterliegt keinen Einschränkungen. Ein Kontext muß nicht explizit erzeugt werden, sondern er entsteht automatisch, sobald ihm eine Zeile zugeordnet wurde.

- add_fix_suggestion($linenumber,$suggestion,$comment)
  Hiermit läßt sich einer Zeile ein Verbesserungsvorschlag zuordnen. Auch hier gilt die Regel, daß mit jedem Aufruf dieser Funktion ein weiterer Vorschlag hinzugefügt wird. Ein solcher Vorschlag besteht in der Regel aus einem Codefragment und einer Erläuterung. Der Parameter '$suggestion' bezeichnet den Code, '$comment' die Erläuterung.

- change_context_criticality($context_name,$severity [,$force_flag])
  Diese Methode kann einem (bereits existierenden) Kontext eine Kritikalität zuordnen. Siehe dazu die Erläuterungen zu 'change_criticality()' oben.

- add_context_annotation($context_name,$text,$severity)
  Die Methode ordnet einem Kontext eine Anmerkungen zu. Siehe dazu die Erläuterungen zu 'add_annotation()' oben.

- add_context_ref($context_name,$ref_name,$reference,$ref_type)
  Die Methode ordnet einem Kontext einen Verweis zu. Siehe dazu die Erläuterungen zu 'add_reference()' oben.

- add_context_fix_suggestion($context_name,$suggestion,$comment)
  Die Methode ordnet einem Kontext einen Verbesserungsvorschlag zu. Siehe dazu die Erläuterungen zu 'add_fix_suggestion()' oben.

- rename_context($old_context_name)
  Es kommt gelegentlich vor, daß man einen Kontext umbenennen muß. Dies ist z.B. der Fall, wenn man einen Kontext mit dem Namen «Accesslist 102» angelegt hat, die Access-Liste aber im Laufe der Konfigurationsbeschreibung redefiniert wird. In diesem Fall kann man den alten Kontext «Accesslist 102» vom System umbenennen lassen. Damit können diesem Kontextnamen

Copyright © Fraunhofer IESE 2003

andere Zeilen zugeordnet werden. Der neue Name des alten Kontextes wird automatisch vom System ermittelt, und als Rückgabewert der Funktion ausgegeben. Somit kann man auch dem «alten» Kontext noch Anmerkungen usw. hinzufügen.

- add_DumpData($name,$position,$text)
  Mit dieser Methode kann man formatierten Text speichern. Jeder Textblock bekommt einen Namen '$name', über den man auf ihn verweisen kann. Die Darstellung des Textes erfolgt als Tabelle. Deshalb muß man bei jedem String, den man in diesen Text-Block einfügen will, die Zeilen- und Spaltennummer angeben, in der der Text erscheinen soll. Möchte man ganzzeilige Ausgaben generieren, kann man anstelle der Spaltennummer auch einen Stern angeben. Die Angabe dieser beiden Werte geschieht mittels des Positions-Strings: Als erstes wird eine Zeilennummer erwartet, dann — durch ein Komma getrennt — die Spaltennummer (oder ein Stern). Somit sieht ein Positions-String, der einen Text in die zweite Spalte der vierten Zeile schreiben will folgendermaßen aus: "4,2". Soll die fünfte Zeile anschließend keine Unterteilung in Spalten besitzen, sondern durchgehend sein (z.B. als Überschrift), so gibt man "5,*" an. Die Positions-Endung ",*" ist optional und darf auch weggelassen werden, d.h. "5,*" ist gleichbedeutend mit "5". Die Spaltengröße der Tabelle richtet sich automatisch nach der Größe der in die Spalte geschriebenen Zeichenketten. Will man zwar eine durchgehende (also spaltenlose) Zeile benutzen, die aber nicht in der ersten, sondern z.B. in der dritten Spalte beginnt, so kann man mittels "6,3*" in der sechsten Zeile ab der dritten Spalte einen Textstring einfügen, der sich dann über alle weiteren Spalten erstreckt und somit auch nicht deren Größe beeinflußt. Dieses kann man z.B. für eine eingerückte Überschrift verwenden. Um mehrere Spalten einer Zeile mit Text zu füllen, muß man die Funktion mehrfach aufrufen. Jeder Dump kann nur von einem Modul bearbeitet werden.

- add_bookmark($name, $ref_name,$reference,$ref_type, [$severity])
  Die Methode ordnet einen Verweis einem Lesezeichen zu. Lesezeichen dienen vor allem als «Aufhänger» für Befunde, die weder zu einer Zeile, noch zu einem Kontext oder einer globalen Befundüberschrift passen. Bei der erweiterten HTML-Ausgabe gibt es in der Mitte des Bildschirmes einen Abschnitt, in dem alle Lesezeichen unter ihrem zugewiesenen Namen '$name' aufgelistet werden. Sie sind nichts anderes als eine Ansammlung von Referenzen, die wie alle anderen Referenzen zu handhaben sind. Siehe dazu die Erläuterungen zu 'add_reference()' oben.

- attach_ciscolinks($linenum, $commandstring [, $context])
  Diese Methode bietet die Möglichkeit, einer Zeile URL-Referenzen auf die Online-Dokumentation von Cisco anzufügen (siehe Abbildung 6, Seite 36, links unten). Die eingefügten Links verweisen auf Syntax- und Funktionsbe-

Copyright © Fraunhofer IESE 2003

schreibungen des Befehls. Als Argument '$commandstring' wird in der Regel einfach der Wortlaut der jeweiligen Konfigurationsklausel angegeben. Die Methode sucht in einer speziell aufbereiteten Datenbank nach passenden URLs für diesen Befehl. Manchmal ist es allerdings nötig, durch die Angabe des Klauselkontexts (d.h. in der Regel: des übergeordneten IOS Configuration Modes) die Suche einzuschränken, damit durch die Suchheuristik nicht zu viele unpassende URLs ermittelt werden. Die URLs für die Befehle finden sich in der Textdatei 'linkdata.conf' im Unterverzeichnis 'Configuration'. Näheres dazu findet sich in Kapitel 7.3.

Copyright © Fraunhofer IESE 2003

Copyright © Fraunhofer IESE 2003

Quellenverzeichnis

CIS	Center for Internet Security (CIS) http://www.cisecurity.org
CiscoTech	Cisco Systems Inc.: Improving Security on Cisco Routers. 1992-2001 http://www.cisco.com/warp/public/707/21.html
IOS12.0a	Cisco Systems Inc.: Cisco IOS 12.0 Network Security. Cisco Press 1999
IOS12.0b	Cisco Systems Inc.: Cisco IOS 12.0 Configuration Fundamentals. Cisco Press 1999
IOSessentials	Cisco Systems Inc.: Essential IOS Features Every ISP Should Consider. Version 2.9, Juni 2001 http://www.cisco.com/public/cons/isp/documents/IOSEssentialsPDF.zip
NSA	U.S. National Security Agency. http://www.nsa.gov
NSAguide	National Security Agency: Router Security Configuration Guide (Version 1.1). NSA Report No. C4-040R-02, September 2002 http://nsa2.www.conxion.com/cisco/download.htm
RAT2.0	Center for Internet Security (CIS): Router Audit Tool. RELEASE 2.0 RC1, January 2003 http://www.cisecurity.org/bench_cisco.html
RFC1918	Network Working Group: Address Allocation for Private Internet. Request for Comments No. 1918, Februar 1996 http://www.faqs.org/rfcs/rfc1918.html
SANS	The SANS (SysAdmin, Audit, Network, Security) Institute http://www.sans.org

Copyright © Fraunhofer IESE 2003

Copyright © Fraunhofer IESE 2003

Liste der Abkürzungen

AAA	Authentication, Authorization, and Accounting
ACL	Access [Control] List
ASCII	American Standard Code for Information Interchange
BGP	Border Gateway Protocol
BNF	Backus-Naur Form
CIS	Center for Internet Security
CSS	Cascading Style Sheet (HTML)
CSV	Comma-separated Vector (Format)
CDP	Cisco Discovery Protocol
CPAN	Comprehensive PERL Archive Network
CROCODILE	Cisco Router Configuration Diligent Evaluator
EIGRP	Extended Interior Gateway Protocol
FTP	File Transfer Protocol
HREF	Hyper(text) Reference (HTML)
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IOS	[Cisco] Internet Operating System
IP	Internet Protocol
ISP	Internet Service Provider
IT	Informationstechnologie
MIB	Management Information Base
MPLS	Multi-Protocol Label Switching
MS	Microsoft
NSA	(US) National Security Agency
NTP	Network Time Protocol
PDF	Portable Document Format
PERL	Practical Extraction and Report Language (bzw.: Pathologically Eclectic Rubbish Lister)
PIX	Private Internet Exchange (Cisco Firewall)
PPP	Point-to-Point Protocol
RADIUS	Remote Authentication Dial-in User Service
RAT	Router Audit Tool
RFC	Request for Comments
SANS	Sysadmin, Audit Network, Security (Institute)
SNMP	Simple Network Management Protocol

Copyright © Fraunhofer IESE 2003

TACACS	Terminal Access Controller Access Control System
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TTY	Teletypewriter (Line Interface)
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VIP	Versatile Interface Processor
VTY	Virtual TTY (Line Interface)
WWW	World-wide Web
XML	Extensible Markup Language

Copyright © Fraunhofer IESE 2003

Document Information

Titel:	CROCODILE Benutzerhandbuch
Datum:	November 2003
Report:	119.03/D
Status:	Version 3.0

Copyright © 2003, Fraunhofer IESE.